High-level Services for Networks-on-Chip

Doctoral Dissertation submitted to the
Faculty of Informatics of the *Università della Svizzera italiana*
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

presented by
Leandro Fiorin

under the supervision of
Prof. Mariagiovanna Sami
co-supervised by
Prof. Cristina Silvano

11 2012

Dissertation Committee

Prof. Matthias Hauswirth	Università della Svizzera italiana, Switzerland
Prof. Laura Pozzi	Università della Svizzera italiana, Switzerland
Prof. Kees Goossens	Eindhoven University of Technology, The Netherlands
Prof. Rainer Leupers	RWTH Aachen University, Germany

Dissertation accepted on 13 11 2012

Prof. Mariagiovanna Sami
Research Advisor
Politecnico di Milano, Italy

Prof. Cristina Silvano
Research Co-Advisor
Politecnico di Milano, Italy

Prof. Antonio Carzaniga
PhD Program Director

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Leandro Fiorin
Lugano, 13 11 2012

To Giovanna and Giancarlo

Considerate la vostra semenza:
fatti non foste a viver come bruti,
ma per seguir virtute e canoscenza.

Dante Alighieri
Divina Commedia, Inferno,
Canto XXVI, 118-120

Abstract

Future technology trends envision that next-generation Multiprocessors Systems-on-Chip (MPSoCs) will be composed of a combination of a large number of processing and storage elements interconnected by complex communication architectures. Communication and interconnection between these basic blocks play a role of crucial importance when the number of these elements increases. Enabling reliable communication channels between cores becomes therefore a challenge for system designers.

Networks-on-Chip (NoCs) appeared as a strategy for connecting and managing the communication between several design elements and IP blocks, as required in complex Systems-on-Chip (SoCs). The topic can be considered as a multidisciplinary synthesis of multiprocessing, parallel computing, networking, and on-chip communication domains.

Networks-on-Chip, in addition to standard communication services, can be employed for providing support for the implementation of system-level services. This dissertation will demonstrate how high-level services can be added to an MPSoC platform by embedding appropriate hardware/software support in the network interfaces (NIs) of the NoC. In this dissertation, the implementation of innovative modules acting in parallel with protocol translation and data transmission in NIs is proposed and evaluated. The modules can support the execution of the high-level services in the NoC at a relatively low cost in terms of area and energy consumption.

Three types of services will be addressed and discussed: *security*, *monitoring*, and *fault tolerance*. With respect to the security aspect, this dissertation will discuss the implementation of an innovative data protection mechanism for detecting and preventing illegal accesses to protected memory blocks and/or memory mapped peripherals. The second aspect will be addressed by proposing the implementation of a monitoring system based on programmable multipurpose monitoring probes aimed at detecting NoC internal events and run-time characteristics. As last topic, new architectural solutions for the design of fault tolerant network interfaces will be presented and discussed.

Acknowledgements

As my Ph.D. studies are finally over, it is time to acknowledge all the people that contributed to this achievement. Even though the journey to reach this goal has been a quite long one, it has given me the possibility to meet lot of interesting people and see lot of new places: in fact, as Don Williams, Jr. once said, *"the road of life twists and turns and no two directions are ever the same. Yet our lessons come from the journey, not the destination*". And now, it comes the moment to thanks all the people that shared with me, for one reason or the other, this journey, hoping not to forget anyone, and kindly asking to be forgiven by those that unwilling will not be mentioned.

I would like to start my acknowledgements thanking Mariagiovanna, that supervised me, with her experience, during my Ph.D. studies. In addition to the pure scientific aspect, I learnt a lot from her: she is an example of passion and dedication to research and scientific life, yet being very approachable and always available despite the high achievements and important awards she obtained during her life. I would like to thanks as well my co-advisor Cristina, which helped Mariagiovanna in providing me with useful comments and research directions during my studies. Despite the small time she had available, she has been always there for discussing new ideas and projects. Moreover, thanks also to the internal and external reviewers of this dissertation, for being available, often at short notice, to read and evaluate my work. I hope that it was not a too hard task.

Thanks also to all my colleagues in ALaRI and Politecnico di Milano: Gianluca for its help and collaboration in the majority of my work, Slobodan for the funny discussions about science, italian language, life and politics, Alberto for its availability and its support in developing new ideas and projects, Francesco for the fights we had and for making me loose lot of time, but also for the useful discussions about security and, in general, research, and (in random order) Daniela, Onur, Marcello, Mauro, Giovanni, Igor, Katerina, Rami, Jelena, Mariano, Marco, Luca for sharing with me the life in the often crowded ALaRI's offices. Thanks also to Umberto, with whom I shared lot of discussions about personal life, projects, administration, etc, most of the time in front of a beer in some "not-so-fancy" pubs somewhere in Europe. And thanks to Janine, Elisa, Danijela, Cristina for their work in Decanato, and for the smile that (most of the time) welcomes me when visiting their office.

A special thank goes also to Prof. Dadda, recently passed away, that represents without any doubt an example of lifelong dedication and passion to research. Even though our main interactions were mainly in me explaining him the use of Winzip or Excel, I can say to have learnt a lot from his figure as researcher, my only regret being not having had the possibility of collaborating with him in his still ongoing research activity.

Last, but not the least, I would like to thank Antonio, that most of everyone else shared with me this part of my life. We have been colleagues, flatmates, drinking buddies, sail mates, wing men, travel buddies, counsellors and confessors, and many other things. Or, in just one single word, friends. Thanks Antonio, without you it would have been for sure much more difficult and boring.

A special thank goes to Hanny. We went together through the majority of these years, until life led us to different paths. Thanks for your support. I learnt a lot also from you.

To conclude, I cannot but thank my family: Floriano, my brother and friend, snowboard addicted, and Violetta, my sister, way wiser than me, even if much younger. Last, but not the least, I want to thank my parents for their continuous support and unconditional love. You are an unrivalled example of hard work and family love, still loving each other as teenagers after 40 years of marriage, and able to raise (and let study) three children with just one worker salary.

Giovanna and Giancarlo, this work is dedicated to you.

Ringraziamenti

Essendo oramai alla fine del mio dottorato, è giunto il momento di ringraziare tutti coloro che hanno contribuito al raggiungimento di questo risultato. Sebbene la strada percorsa per arrivare a questo punto sia stata lunga, mi ha dato la possibilità di incontrare una gran quantità di gente interessante e visitare diversi luoghi mai visti prima: come disse infatti Don Williams, Jr., "*La strada della vita si intreccia e curva, e direzioni diverse conducono a finali sempre diversi. Eppure, gli insegnamenti ci arrivano dal viaggio, e non dalla destinazione*". A questo punto è arrivato il momento di ringraziare tutti coloro che, per una ragione o l'altra, hanno condiviso con me questo viaggio, sperando di non dimenticare nessuno e chiedendo a coloro che per errore non verranno nominati di scusarmi.

Vorrei cominciare ringraziando Mariagiovanna, che mi ha seguito e consigliato durante i miei studi di dottorato. Oltre all'aspetto puramente scientifico, ho imparato molto da lei: Mariagiovanna è un esempio di passione e dedizione alla ricerca e alla scienza, e, nonostante gli importanti riconoscimenti ricevuti e gli importanti risultati ottenute nella sua carriera scientifica, una persona alla mano e sempre disponibile. Vorrei inoltre ringraziare Cristina, che ha aiutato Mariagiovanna nel suo lavoro come advisor fornendomi commenti e direzioni di ricerca durante i miei studi. Nonostante il poco tempo a disposizione, Cristina è sempre stata disponibile a discutere con me nuove idee e progetti. Inoltre, vorrei ringraziare i revisori interni ed esterni per essere stati disponibili, spesso con poco preavviso, a leggere e valutare il mio lavoro. Spero che l'impegno non sia stato troppo gravoso.

Grazie anche ai miei colleghi in ALaRI e al Politecnico di Milano: Gianluca, per il suo aiuto e la sua collaborazione durante la maggior parte del mio lavoro, Slobodan, per le divertenti discussioni a proposito di scienza, lingua italiana, vita e politica, Alberto, per la sua disponibilità e il suo aiuto nello sviluppare nuove idee e progetti, Francesco, per le lunghe discussioni ed il tempo che mi ha fatto perdere, ma anche per le utili discussioni riguardati la sicurezza e la ricerca in generale, e (in ordine causale) Daniela, Onur, Marcello, Mauro, Giovanni, Igor, Katerina, Rami, Jelena, Mariano, Marco, Luca, per condividere con me il tempo speso negli spesso affollati uffici di ALaRI. Grazie anche ad Umberto, con il quale ho condiviso una gran quantità di discussioni su vita, progetti, amministrazione, etc, il più delle volte di fronte ad una birra in qualche locale non proprio elegante da qualche parte in Europa. E grazie anche a

Janine, Elisa, Danijela, Cristina per il loro lavoro al Decanato e per il sorriso che (il più delle volte) mi accoglie quando vado a trovarle in ufficio.

Un ringraziamento speciale va anche a Prof. Dadda, recentemente mancato, che rappresenta senza alcun dubbio un esempio di vita dedita con passione alla ricerca. Sebbene abbia interagito con lui solo per spiegargli il funzionamento di Winzip o Excel, posso sinceramente affermare di aver comunque imparato molto dalla sua figura di ricercatore. Il mio unico dispiacere è stato quello di non aver avuto la possibilità di collaborare con lui nella sua ricerca, ancora portata avanti nonostante i suoi 86 anni.

Per ultimo vorrei ringraziare Antonio, che più di ogni altro ha condiviso con me questa parte della mia vita. Siamo stati colleghi, coinquilini, compagni di bevute, veleggiata, uscite e viaggi, consiglieri e confessori, e molto altro. O, in una sola parola, amici. Grazie Antonio, senza di te sarebbe stato molto più difficile e noioso.

Un ringraziamento speciale va poi a Hanny. Abbiamo passato insieme la maggior parte di questi anni, fino a quando la vita ci ha condotto su percorsi differenti. Grazie per il tuo sostegno. Ho imparato tanto anche da te.

Per concludere, non posso che ringraziare la mia famiglia. Floriano, mio fratello, amico e drogato di snowboard, e Violetta, mia sorella, molto più saggia di me nonostante la più giovane età. Per ultimi, ma non meno importanti, vorrei ringraziare i miei genitori per il loro continuo sostegno per il loro amore incondizionato. Siete per me un esempio inarrivabile di lavoro duro e dedizione alla famiglia: dopo 40 anni di matrimonio vi amate ancora come ragazzini, e siete stati capaci di crescere (e far studiare) tre figli con un solo stipendio da operaio.

Giovanna e Giancarlo, questo lavoro è dedicato a voi.

Contents

Figures

Tables

Chapter 1

Introduction

Future technology trends [2] envision that next generation Multiprocessors Systems-on-Chip (MPSoCs) will be composed of a combination of a high number of processing and storage elements interconnected by complex communication architectures. Processing elements will include general purpose processors and specialized cores, such as digital signal processors (DSPs), very long instruction word (VLIW) cores, programmable cores (FPGAs), application specific processors, analog front-end devices, and peripheral devices. The integration of many heterogeneous cores will be possible thanks to advances in nanoelectronic technologies, enabling billions of transistors on a single chip, running at multi-GHz speed, and operating at supply voltages below one volt.

As an example of this trend, an integrated architecture containing 80 tiles arranged as an 8 x 10 2-D array of floating-point cores and interconnect elements was recently presented by Intel [3]. Each tile has two pipelined single-precision floating-point multiply accumulators (FPMAC) which feature a single-cycle accumulation loop for high throughput, and a packet-switched router. Both processing elements and interconnect are designed to operate at 4 GHz, providing a bisection bandwidth of 2 Terabits/s.

Another example is given by the Intel Single-Chip Cloud Computer project [4], which incorporates 48 Itanium cores organized in 24 tiles, each of them containing two cores. Tiles are connected through a mesh network of 24 routers, with bisection bandwidth of 256 GB/s.

In such a type of system, one solution to reduce design complexity, as well as the gap between advances in the development of manufacturing technology and those of synthesis and compiler technology, has been foreseen in the reuse of previously developed Intellectual Propriety cores (IPs), that will constitute the basic blocks for the designers. IP cores used in an individual MPSoC need not be homogeneous - actually, adopting heterogeneous cores may lead to relevant benefits in cost and performances when application-targeted embedded systems are considered. However, the design of this type of heterogeneous systems raises the problem of having tools for synthesis and

compilation that could efficiently exploit the large number of resources available to designers.

Communication and interconnection between these basic blocks play crucial roles when the number of such blocks increases. Indeed, the complexity of the new systems spawns the challenge of enabling reliable communication channels between cores; a challenge that becomes more and more difficult as the number of integrated cores per design increases. Therefore, the traditional solution for inter-core communication based on a shared bus, very soon becomes unable to guarantee sufficient levels of efficiency, both from the performance and the power consumption perspectives.

Moreover, the distribution of the global wiring in the System-on-Chip generates several problems related to the physical limitations of interconnect implementation with new deep-submicron integrated technologies. As dimensions of wirings decrease - following technology trends in device shrinking - the interconnection delay is likely to largely exceed the clock period [5]. The wire delay δ can be in general expressed as proportional to $C \cdot R$, where C is the capacitance of the wire, and R is its resistance. By reducing the geometries of wires by a factor S, while the capacitance remains constant, the resistance of the wires increases by a factor S^2, increasing as well by the same factor the wire delay. Therefore, technology trends aiming at obtaining faster and faster cells bring as side effect slower and slower interconnection lines, as well as a higher relative importance of the spreading of physical parameters (e.g. variance of wire delay per unit length) when compared to the timing reference signal (e.g. clock period). All these facts make it more difficult to safely meet signal timing constraints [5].

Other problems related to current technology trends should also be considered due to their influence on the design of the communication infrastructure. In particular, it will be increasingly difficult to guarantee the integrity of signals transmitted over the wires. The reduced *signal swings*, and the corresponding reduction in the voltage noise margin, adopted in on-chip communications to limit in the device's power consumption, will make it increasingly possible to receive signals corrupted by some external sources of noise, such as for instance those due to electromagnetic interferences, or by soft errors caused by the collision of thermal neutrons (due to the decay of cosmic ray showers) and/or alpha particles (due to impurities in the package). At the same time, reduction of the space between wires observed when shrinking feature sizes of integrated devices implies a large increase in *crosstalk*, increasing therefore the possible sources of noise influencing the transmission of signals.

In order to overcome all the above technological problems, Networks-on-Chip (NoCs) [6, 7] appeared as a strategy to manage the communication between several design elements and IP blocks, as required in complex Systems-on-Chip (SoCs), allowing the transition from a *core-centric* design approach to a *communication-centric* design approach [7]. The topic can be considered as a multidisciplinary synthesis of multiprocessing, parallel computing, networking, and on-chip communication domain. NoCs are usually composed of *network interfaces* (NIs), that adapt data and control signals

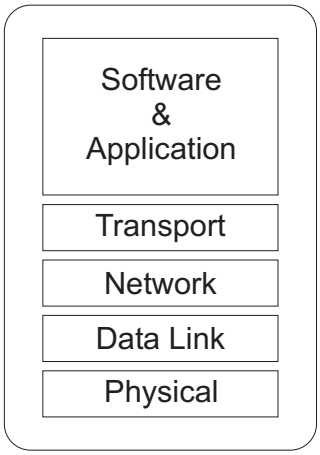


Figure 1.1. NoC micro-network stack.

received by cores to the packetized communication protocol employed in the NoC, and *routers*, that route packets to their destination node in the NoC.

As in wide-area networks, a layered-stack approach has been proposed for an efficient design of on-chip inter-core communication medium, exploiting the lesson learned by telecommunication community. Similarly to the ISO-OSI Reference Model, the global on-chip communication is decomposed into several layers [7], in order to provide the designers with a clear interface between the functionalities that such a system should offer.

Figure 1.1 shows the NoC micro-network stack [7]. The *Physical Layer* is related to all the aspects concerning the transmission of signals through physical wires, including techniques and circuits for driving information. As previously discussed, many technology problems are related to this layer, due to the increased noise sensitivity in signal transmission brought by the shrinking of integrated devices. In on-chip networks, signals may be allowed to arrive corrupted at the destination, leaving to higher layers in the stack to take care of corrections and reliability of the overall communication.

As an example of this trend, packetized transmissions and *error correcting codes* (ECCs) are implemented in the *Data Layer*, in order to retrieve and correct faulty signal transmissions. The Data Layer is in fact in charge of ensuring, up to a minimum required level, a reliable data transfer despite of the physical unreliability of links, and of dealing with contention by multiple IPs of the shared resources of the communication medium.

The *Network Layer* implements end-to-end delivery control, setting up connections between successive links and routing information from sources to destinations. Three main type of switching techniques can be identified to describe connections set up:

- With the *Store-and-Forward* technique, at each intermediate step along the routing path the packet is stored in a buffer, waiting for the whole packet to be

completely received; only at that moment the packet is forwarded to the next step of the path. While allowing the implementation of more elaborated routing schemes, the Store-and-Forward technique requires to have in each router in the path enough buffer resources to store all the complete packets that may simultaneously arrive.

- In the *Virtual-Cut-Through* scheme, packets are forwarded as soon as received, reducing therefore the delay due to storing the packet before forwarding it. Buffering resources needed are still significant, due to the fact that entire packets have to be stored in intermediate steps if resources for the following steps are not available.

- In *Wormhole* switching, packets are divided into smaller pieces called flits (flow control digits). The size of the flits is usually equal to the width of the physical link. The header flit, containing routing information, reserves resources in routers along the path. Body flits follow the header flit, while the tail one releases the resources. Flits are forwarded as soon as received by the routers, without storing the whole packet as in the previous schemes. The packet's last flit releases resources for subsequent communications. While in Wormhole switching buffering resources are smaller compared to the two previous schemes, problems due to *deadlock* and *livelock* may arise [5].

Concerning routing strategies that can be adopted, it is possible to distinguish between *deterministic* routing, where the same path is associated with the same source-destination pair, and *adaptive* routing, where paths can be modified according to information such as traffic congestion, power consumption, and reliability of physical links.

The *Transport Layer* manages end-to-end services and deals with packets segmentation and assembly. Unlike in data networks, packet sizes can be customized to application requirements, in order to better satisfy the application needs in terms of energy consumption, traffic congestions, or latency of the communication.

Upper layers can be viewed as merged in a generic *Software and Application Layer*, which includes system and application software. System software provide an abstraction of the underlying hardware platform. Application software exploit the intrinsic parallelism of NoC architectures, taking into account the communication aspects.

Even if the NoC idea seems an adaptation to the SoC context of similar computer networks concepts, many research issues are still open, due to their different constraints and amount of resources available in the two cases. In fact, some main unique NoCs features are: the spatial locality of modules connected, the reduced non-determinism of on-chip traffic, the constraints in terms of energy consumption and low latency, the possibility of applying specific stack services, and the need for low cost solutions.

1.1 Dissertation contributions

The main goal of this dissertation is to demonstrate how high-level services can be added to an MPSoC platform by embedding appropriate hardware/software support in the network interfaces of the NoC. To achieve this goal, in this dissertation we propose and evaluate the implementation of innovative modules, that, by acting in parallel with protocol translation and data transmission in the NI, can support the execution of high-level services in the NoC at a relatively low cost in terms of area and energy consumption.

In particular, we target *security*, *monitoring*, and *fault tolerance* in NoCs:

- **Security:** as computing and communications increasingly pervade our lives, security is gaining an increasing relevance in the development of new electronic devices. This fact is especially true in the case of embedded systems, intrinsically constrained in terms of computational capacity, storage size, and energy availability. In the case of NoCs, the design of secure architectures has been taken into account only recently. In this dissertation, we therefore address the security aspects related to NoC-based platforms. In particular, the main contributions of this dissertation to the security topic mainly focus on the introduction of innovative mechanisms for providing support for data protection and run-time detection of malicious attacks. The proposed mechanisms are based on the use of dedicated hardware modules embedded within the NI. The modules guarantee secure accesses to memory and/or memory-mapped peripherals by enforcing access control rules specifying the way in which an IP initiating a transaction to a shared memory in the NoC can access a memory block. Moreover, as part of a secure monitoring system, they can be employed for detecting attempts of illegal access to protected memory blocks as well as Denial-of-Service attacks.

- **NoC monitoring:** the complexity of future NoC-based MPSoC platforms raises the problem of exploiting efficiently the amount of resources available, and of understanding the system behavior once the platform has been implemented. For NoCs, new tools are needed for helping designers in these tasks, exploiting information derived by measurements taken on the running system. On this topic, the main contributions of this dissertation focus on the idea of using the NI to monitor run-time system behavior by tracing communication activities, in order to retrieve information useful for performance analysis, run-time optimization, and optimal resources allocation. In particular, we detail the implementation of a programmable multipurpose probe, that, embedded within the NI, can provide to a central management unit information about throughput and latency of NoC transactions, as well as utilization of buffers in NI and routers.

- **Fault tolerance:** the aspects of fault tolerance in NoC-based architectures play a role of increasing relevance, as complexity of the design increases, and as CMOS

technology scales down into the deep-submicron domain. In fact, devices and interconnect are subjected to new types of malfunctions and failures that are hard to predict and avoid with the current design methodologies. Fault tolerant approaches are therefore necessary to overcome these limitations, and new methodologies and architectural solutions should be explored. With respect to the fault tolerance topic, the main contributions of this dissertation focus on the analysis of the fault susceptibility of NoC components, and on the proposal and evaluation of a fault tolerant architecture for the NI, which represents the most critical component from the point of view of the resistance to faults. The solution proposed, based on the use of a combination of error detecting and correcting codes and a limited amount of architectural redundancy, allows a reduction of the design costs with respect to standard fault tolerant techniques based on triple modular redundancy, by maintaining similar levels of robustness to faults.

1.2 Organization of the document

The remainder of this document is organized as follows. Chapter 2 describes the characteristics of the NoC-based MPSoC platform taken as reference in the following chapters of the dissertation. Chapter 3 discusses the motivations for implementing the high-level services studied in this work. In particular, section 3.1, presents an overview of security threats that may affect NoCs, as well as discussing the need in NoC-based architecture of implementing secure architectures to prevent attacks aiming at accessing critical information or disrupting system services. Section 3.2 discusses motivations for implementing a monitoring system in the NoC that could help significantly both in understanding at design time the platform behavior, and in optimizing at run-time resource utilization, while section 3.3 demonstrates that network interfaces are critical elements from the point of view of the fault tolerance of the overall NoC, and that appropriate methodologies should be applied for their design. In chapter 4, related work is discussed, in particular focusing on the three main services targeted in this dissertation. Chapter 5 presents the main contributions of this dissertation on the aspect of security: it describes and evaluate a data protection mechanisms for preventing illegal accesses to protected memory blocks in NoC-based architecture, as well as a secure monitoring system for detecting attempts of illegal access to protected memory blocks and Denial-of-Service attacks. Chapter 6 deals with the aspect of NoC monitoring, by detailing the implementation of the several components of the monitoring system and evaluating their implementation costs and overhead. Chapter 7 presents the work performed towards the goal of implementing a fault tolerant NI, as well as discussing implementation costs and resilience to faults of the proposed architecture. Finally, chapter 8 summarizes achieved results and discusses future work.

Chapter 2

Reference MPSoC architecture

This chapter introduces the NoC-based Multiprocessor System-on-Chip architecture taken as reference in this dissertation. The reference platform is a shared-memory heterogeneous architecture composed of general purpose and custom dedicated processors and cores (such as DSP, VLIW, and FPGA cores), storage elements and peripherals. Cores in the platform, heterogeneous for functionalities offered and characteristics, communicate through an NoC. Signals coming from the communication interfaces of the cores are translated by the network interface (NI) into packets compliant with the protocol used within the NoC. We will assume a network interface compliant with the specifications of the Open Core Protocol (OCP) interface [8]. The main characteristics of the OCP interface will be described in better detail in section 2.1.1

The communication is transaction-based in order to present IP modules with a shared-memory abstraction [9]. As a consequence, elements on the NoC are memory mapped. Following the transaction-based protocol, it is possible to distinguish between IP modules acting as *initiators* or as *targets*. *Initiators* are enabled to begin both a *load (read)* and a *store (write)* transaction to *targets*. The protocol used to exchange data between the initiator and the target in the network is simple and always requires to send back an acknowledgement (or not acknowledgement) message to the initiator for each request, both for *load* and *store* transactions. If the transaction is accepted, the target replies by sending back an acknowledgement message (*ack*) and, in the case of a *load*, also the requested data. If the transaction is rejected, a not acknowledgement message (*nack*) is sent back to the initiator. A description of the main elements of the NoC, i.e. the network interface and the router, is given next.

2.1 Reference network interface

The network interface translates IP signals and commands into packets, compliant with the NoC protocol. Figure 2.1 shows the basic functional blocks of the network interface taken as reference point in this dissertation. In general, a NI includes a front-end and a

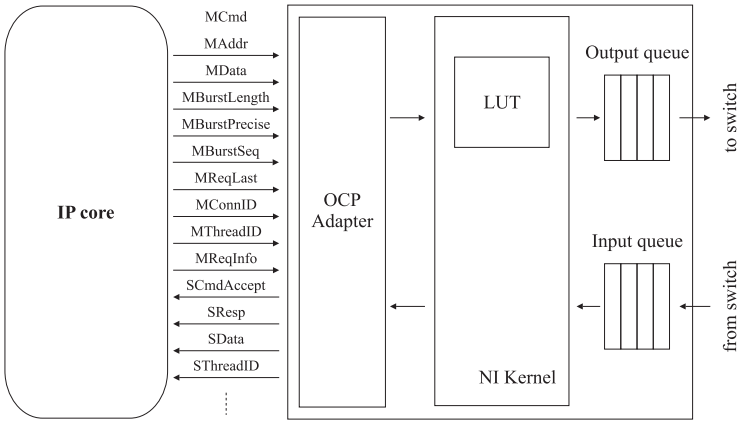


Figure 2.1. Overview of the reference NI architecture.

back-end sub-module [5, 10, 9]. The front-end module implements the communication protocol adopted by the core. The back-end module is in charge of implementing basic communication services, such as data packetization, and routing and control flow related functions. Moreover, additional services, such as link error detection and error recovering strategies, transactions ordering, support for cache coherence and security, can also be implemented [5].

Several alternatives are available for the implementation of the basic services provided by the NI, and in particular for the packetization phase [5]. In this work, we will consider the NI as an independent hardware block located between the core and the communication infrastructure. The main NI components considered are:

- An OCP [8] adapter, which implements an instance of the OCP interface. At cores acting as initiators, the adapter implements a *Slave* interface, while at target cores, it implements a *Master* interface.

- The NI kernel, which receives and transmits data and control information from/to the adapter, packetizes and de-packetizes messages, schedules and inserts packets in the output queue and retrieves them from the input queue, and finally implements the control flow mechanism.

- The input and output FIFO queues, which on the producer side store packets ready to be inserted into the NoC, and on the receiver side store incoming packets.

When a new transaction is requested by the processing element, the NI lookups the memory-mapped address of the OCP transaction by employing a programmable lookup table (LUT), located into the NI kernel [10]. The LUT returns as output the routing information to be inserted into the packet header. At each router encountered along the path to the destination node, the routing information is checked for requesting the

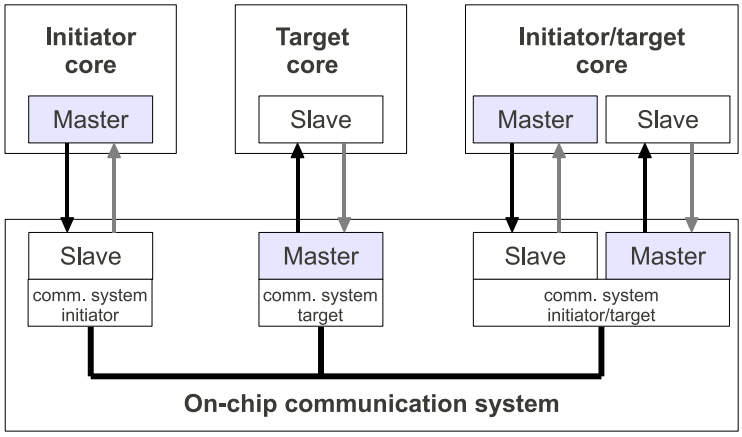


Figure 2.2. System showing OCP instances and interfaces.

desired output port. The LUT is programmable in the sense that information stored can be rewritten to support run-time modifications of the routing paths, caused for instance by the presence of faulty links in the NoC. The NoC implements a wormhole flow-control. Packets generated by the NI are therefore divided in subsequent flits.

2.1.1 OCP interface

The Open Core Protocol (OCP) is a protocol that comprehensively describes the system level integration requirements of intellectual property (IP) cores [8]. The goal of the OCP specifications is to clearly define design boundaries between cores, in order to enable an independent design of the cores of the system, and to support the reuse of IP cores as well as the reuse of verification and test suites. OCP interfaces can be customized by designers in order to match core and platform requirements: both simple interfaces for low-performance peripheral cores and complex interfaces for high-performance microprocessors can be implemented. The OCP defines a point-to-point interface between two communicating entities. One of the entities acts as master of the communication. The other entity acts as slave. The master initiates and controls the communication, while the slave responds to the commands issued by the master, either by accepting data from it, or giving data to it. As an example, figure 2.2 shows a simple system containing a OCP-wrapped communication subsystem and three IP core entities, acting as system initiator, system target, and both, respectively.

In this dissertation, we refer to a NI compliant with the OCP specifications. The main OCP signals that will be used in this work for specifying the interface of a processing element acting as initiator are the following (figure 2.1):

- **Clk** - clock signal;
- **Reset_N** - reset signal;

- **MCmd** - OCP transfer type requested by the master, i.e., either a read or write type request;
- **MAddr** - slave-dependent address of the resource targeted by the transfer;
- **MData** - data carried from the master to the slave in the case of a write operation;
- **MDataValid** - valid write data;
- **MRespAccept** - master accepts response;
- **MBurstLength** - number of transfers in a burst;
- **MBurstPrecise** - field indicating whether the precise length of a burst is known at the start of the burst or not;
- **MBurstSeq** - signal specifying the sequence of addresses for requests in a burst;
- **MReqLast** - it specifies the last request in a burst;
- **MConnID** - connection identifier;
- **MThreadID** - thread identifier associated with the current transfer request;
- **MReqInfo** - this field can be used to send additional information sequenced with the request;
- **SCmdAccept** - slave accepts transfer;
- **SResp** - response field from the slave to a transfer request from the master;
- **SData** - data carried from the slave to the master in the case of a read request;
- **SReqLast** - it specifies the last response in a burst;
- **SThreadID** - response thread identifier.

As discussed, the OCP interface can be customized by adding or removing some of the signals, depending on the functionalities implemented and on the type and complexity of the core.

2.2 Reference router

In NoCs, routers are in charge of directing to their destination packets injected by NIs into the on-chip communication system. Several architectures have been proposed for the implementation of the router. Without loss of generality, in this dissertation we refer to the basic router architecture shown in figure 2.3. In the figure, *Input Buffers* are

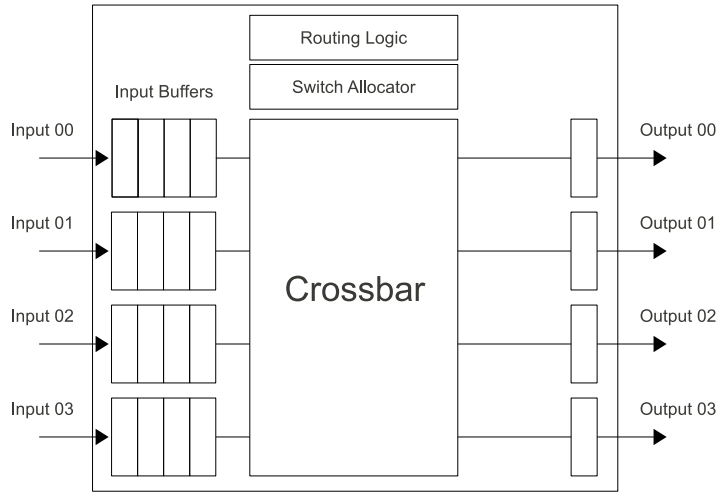


Figure 2.3. Overview of the reference router architecture considered in the experiments.

employed at the router's input ports for temporarily storing flits coming from previous connections, while waiting for the destination ports to be available for the transmission. The *Routing Logic* reads the header of the packet (usually contained in the first flit of the packet) and requests the utilization of the desired output ports. Several implementations exist for the routing logic, depending on the routing strategy adopted in the NoC. In the case of *distributed routing*, each packet carries the destination address. At each router this information is processed through the use of lookup tables or circuit logic implementing the routing decision algorithm, in order to define the desired router's output port. In the case of *source routing*, the packet header contains the routing choice about the output port to request in each hop of the path to destination. Information is pre-calculated and stored into the NI, which inserts the routing information into the header at the creation of each new packet. Moreover, routing can be *static* or *dynamic*, depending on the possibility to adapt at run-time at the state of the network. For each router's output port, the *Switch Allocator* selects the input port allowed to use it, arbitrating its utilization among the input ports requesting it, and selecting the right configuration in the connections of the *Crossbar*.

In this dissertation, we will refer to input buffered router architectures which implement a *static* routing. Depending on the topic, in our experiments we will both consider a *distributed* (table-based) routing (in chapter 5 and chapter 6) and a *source* routing (in chapter 7). Solutions proposed can be however applied to both routing strategies.

Chapter 3

Motivations

This dissertation focuses on the idea of providing Networks-on-Chip with high-level functional services. These services add new functionalities to the system, and their implementation depends on the application needs and on the type of platform employed in the design [5], as well as on the possibility of finding the right trade-off between the offered services and the related implementation costs. In principle, additional services can be implemented in different alternative ways: via software, dedicated hardware, or processor modifications [5]. The main idea presented in this dissertation is to implement services within network interfaces, on top of the standard communication services usually provided by them. High level services will be implemented as additional modules running in parallel to the normal NI activity, in order to avoid degradation of the performances.

This chapter presents motivations behind the introduction of each functional services proposed and discussed in this dissertation. More in detail, in section 3.1, different possible attacks to NoC MPSoCs are described, which motivate the introduction of modules to support security in the system. In particular, section 3.1.3 presents a preliminary study of the effects of *Denial-of-Service* (DoS) attacks on NoCs. Section 3.2 discusses motivations for adopting a monitoring service in the NI. Section 3.3 presents a study about the susceptibility to faults of the NI, which justifies the introduction of fault tolerance techniques in its implementation.

3.1 The need for security-aware NoCs

As computing and communications increasingly pervade our lives, security and protection of sensitive data and systems are emerging as extremely important issues. This is especially true for embedded systems, often operating in non-secure environments while at the same time being constrained by such factors as computational capacity of microprocessor cores, memory size, and in particular power consumption [11, 12, 13, 14]. Due to such limitations, security solutions designed for general pur-

pose computing are not suitable for this type of systems. At the same time, viruses and worms for mobile phones have been reported since several years [15], and they are foreseen to develop and spread as the targeted systems will increase in functionalities offered and in complexity. Currently known malware are able to spread through Bluetooth connections or MMS (Multimedia Messaging Service) messages and infect recipients' mobile phones with copies of the virus or worm, hidden under the appearance of common multimedia files [16].

In the context of the overall embedded SoC/device security, security-awareness is therefore becoming a fundamental concept to be considered at each level of the design of future systems, which should be included as good engineering practice since the early stages of the design of software and hardware platforms. Networks-on-Chips should be considered in the secure-aware design process as well. In fact, the advantages in terms of scalability, efficiency and reliability given by the use of such a complex communication infrastructure may lead to new weaknesses in the system that can be critical and should be carefully studied and evaluated. On the other hand, NoCs can contribute to the overall security of the system by providing an additional mean to monitor system behavior and detect specific attacks [17, 18]. Communication architectures can effectively react to security attacks by disallowing the offending communication transactions, or by notifying appropriate components of security violations [19]. The particular characteristics of NoC architectures make it necessary to afford the security problem in a comprehensive way, encompassing all the various aspects from silicon-related ones to network-specific ones, both with respect to the families of attacks that should be expected and to the protective countermeasures that must be created.

In this dissertation, we will mainly focus on two aspects that can be considered critical for enhancing security in future NoC-based architectures [11, 12], namely: *protecting critical data in shared memory MPSoCs* and *monitoring of system activities for security purposes*. Motivations for focusing on these two aspects are presented in subsection 3.1.2 and in subsection 3.1.3. We start discussing the need of security by presenting a taxonomy of security threats affecting NoCs, and, more generally, embedded systems.

3.1.1 Attack taxonomy

Adding specific security features to a system implies additional costs in the design as well as during the lifetime of the devices, respectively in terms of modifications in design flow and in the need of additional hardware and software modules, as well as in performance and power consumption increase [11]. Therefore, it is mandatory to understand the requirements in terms of security of the system, i.e., which security violation it will be able to efficiently counteract. This subsection overviews typical attacks that could be carried out against an embedded system, providing a classification in terms of the agent used to perform the attack and of its targets. The subsection dis-

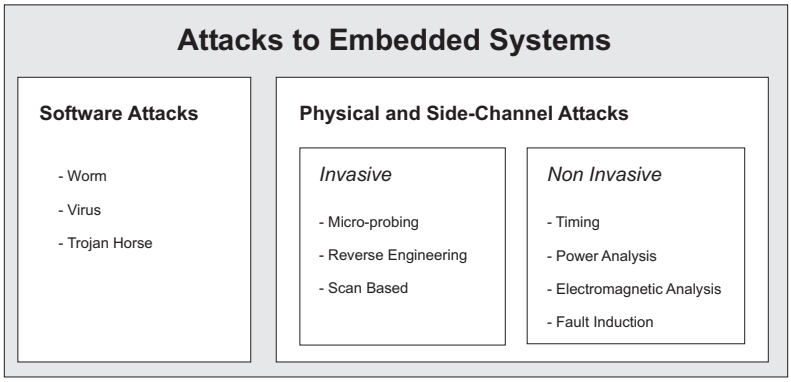


Figure 3.1. Attacks to embedded systems.

cusses various types of security threats, namely those exploiting software, physical and invasive techniques, and side channels techniques. After reviewing the most relevant types of attacks brought against SoCs "in general", special attention will be given to those that may exploit the intrinsic characteristics of the communication system in a System-on-Chip based on an NoC.

Attacks addressing SoCs

Figure 3.1 shows a possible classification of the attacks in general addressing embedded systems [20]. The given classification is based on the type of agent used to perform the attacks. One or more types of agents can be employed by a malicious entity attempting to achieve its objectives on the addressed system, and they can cause problems in terms of privacy of information, integrity of data and code, and availability of the system's functionalities.

Software attacks

Software attacks exploit weaknesses in system architecture, through malicious software agents such as *virus*, *trojan horses*, *worms*, etc. These attacks address pitfalls or bugs in code, such as in the case of attacks exploiting buffer overflow or similar techniques [21]. As embedded systems software increases in complexity and functionalities offered, they are foreseen to become an ideal target for attacks exploiting software agents. Viruses for mobile phones have been reported since several years [15], and similar attacks are likely to be extended to embedded devices in automotive electronics, domotic applications, networked sensors and more generic pervasive applications. Due to the cheap and easy infrastructure needed by the hacker to perform a malicious task, software attacks represent the most common source of attack and the major threat to face in the challenge to secure an embedded system. Moreover, while increasing the flexibility of the system, the possibility of updating functionalities

and downloading new software applications increases also its vulnerability to external attackers and maliciously crafted applications' extensions. An additional challenge is also represented by the extended connectivity of embedded devices [12], which implies an increase in the number of security threats that may target the system, physical connections to access the device not being anymore required.

Typical embedded system viruses will spread through the wireless communication channels offered by the device (such as Bluetooth) and install themselves in unused space in Flash ROM and EEPROM memories, immune to re-booting and re-installation of the system software. Malicious software is in this way almost not visible to other applications on the system, and capable of disabling selected applications, including those needed to disinfect it [22].

Physical attacks

Physical attacks require physical intrusion into the system at some levels, in order to directly access the information stored or flowing in the device, modify it or interfere with it. These types of attack exploit the characteristic implementation of the system or some of its properties to break the security of the device. The literature usually classifies them as *invasive* and *non-invasive* [23].

Invasive attacks require direct access to the internal components of the system. For a system implemented on a circuit board, inter-component communication can be eavesdropped by means of probes to retrieve the desired information [11]. In the case of Systems-on-Chip, access to the internal information of the chip implies the use of sophisticated techniques to depackage it and the use of micro-probes to observe internal structure and detect values on buses, memories and interfaces. A typical *micro-probing* attack would employ a *probing station*, used in manufacturing industry for manual testing of product line samples, and consisting of a microscope and micromanipulators for positioning microprobes on the surface of the chip. After depackaging the chip by dissolving the resin covering the silicon, the layout is reconstructed using in combination the microscope and the removal of the covering layers, inferring at various level of granularity the internal structure of the chip. Microprobes or e-beam microscopy are therefore used to observe values inside the chip. The cost of the infrastructure makes microprobing attacks difficult to be used. However, they can be employed to gather information on some sample devices (e.g., information on the floorplan of the chip and the distribution of its main components) that can be used to perform other types of non-invasive attacks.

Non-invasive attacks exploit externally available information, unintentionally leaking from the observed system. Unlike invasive attacks, the device is not opened or damaged during the attack. There are several types of non-invasive attacks, exploiting different sources of information gained from the physical implementation of a system, such as power consumption, timing information, or electromagnetic leaks.

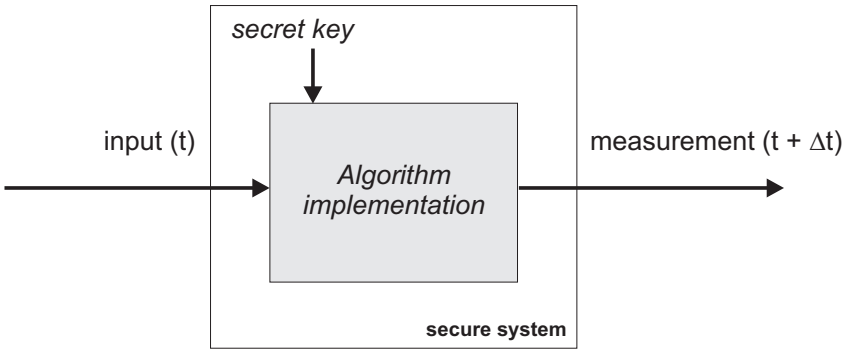


Figure 3.2. Representation of the timing attack.

Timing attacks were first introduced by Kocher [24]. Figure 6.4 shows a representation of a timing attack. The attacker knows the algorithm implementation and has access to measurements of the inputs and of the outputs of the secure system. The attacker's goal is to discover the *secret key* stored inside the secure system. Starting point for the attacker is the observation that the execution time of computations is data-dependent, and hence secret information can be inferred from its measurement. In such attacks, the attacker observes the time required by the device to process a set of known inputs with the goal of recovering a secret parameter (for instance the cryptographic key inside a *smart-card*). The execution time for hardware blocks implementing cryptographic algorithms depends usually on the number of '1' bits in the key. While the number of '1' bits alone is not enough to recover the key, repeated executions with the same key and different inputs can be used to perform statistical correlation analysis of timing information and therefore recover the key completely. Delaying computations in order to make them multiple of the same amount of time, or adding random noise or delays increases the number of measurements required, but do not prevent the attack. Techniques exist however to counteract timing attacks at the physical, technological or algorithmic level [23].

Power analysis attacks [24] are based on the analysis of the power consumption of the device while performing the encryption operation. Main contributions to power consumption are due to gate switching activity and to the parasitic capacitance of the interconnect wires. The current absorbed by the device is measured by very simple means. It is possible to distinguish between two type of power analysis attacks: *simple power analysis* (SPA) and *differential power analysis* (DPA).

SPA involves direct interpretation of power consumption measurements collected during cryptographic operations. Observing the system's power consumption allows identifying sequences of instructions executed by the attacked microprocessor to perform a cryptographic algorithm. In those implementations of the algorithm in which the execution path depends on the data being processed, SPA can be directly used to interpreter the cryptographic key employed. As an example, SPA can be used to break

implementations of the public-key cryptography algorithm RSA by revealing differences between multiplication and squaring operation performed during the modular exponentiation operation [24]. If the squaring operation is implemented (due to code optimization choices) differently than the multiplication, two distinct consumption patterns will be associated with the two operations making it easier to correlate the power trace of the execution of the exponentiator to the exponent's value. Moreover, in many cases SPA attacks can help to reduce the search space for brute-force attacks. Avoiding procedures that use secret intermediates or keys for conditional branching operations will help to protect against this type of attack [24].

DPA attacks are harder to prevent. In addition to the large-scale power variations used in SPA, DPA exploits the correlation between the data values manipulated and the variation in power consumption. In fact, it allows adversaries to retrieve extremely weak signals form noisy sample data, often without knowing the design of the target system. To achieve this goal, these attacks use statistical analysis and error-correction statistical methods to gain information about the key. The power consumption of the target device is repeatedly and extensively sampled during the execution of the cryptographic computations. Goal of the attacker is to find the secret key used to cypher the data at the input of the device, by making guesses on a subset of the key to be discovered, and calculating the values of the processed data in the point of the cryptographic algorithm selected for the attack. Power traces are collected and divided into two subsets, depending on the value predicted for the bit selected. The differential trace, calculated as the difference between the average trace of each subset, shows spikes in regions where the computed value is correlated to the values being processed. The correct value of the key can thus be identified from the spikes in its differential trace. As an example, figure 3.3 shows a simulation of DPA attack on a Kasumi S-box implemented in CMOS technology [1]. The Kasumi block cipher is a Feistel cipher with eight rounds, with a 64-bit input and a 64-bit output, and a secret key with a length of 128 bits. Kasumi is used as standardized confidentiality algorithm in 3GPP (3rd Generation Partnership Project) [25]. In figure 3.3 it is possible to note how the differential trace of the correct key (plotted in black) presents the highest peak, being therefore clearly distinguishable from the remaining ones and showing a clear correlation to the values processed by the block cipher. For a more detailed discussion of DPA attacks, see [24].

Electromagnetic analysis (EMA) attacks exploit measurements of the electromagnetic radiations emitted by a device to reveal sensitive information. This can be performed by placing coils in the neighborhood of the chip and studying the measured electromagnetic field. The information collected can be therefore analyzed with simple analysis (SEMA) and differential analysis (DEMA) or more advanced correlation attacks. Compared to power analysis attacks, EMA attacks present a much more flexible and challenging measurement phase (in some cases measurement can be carried out at significant distance from the device - 15 feet [23]), and the provided information offers a wide spectrum of potential information. A deep knowledge of the layout

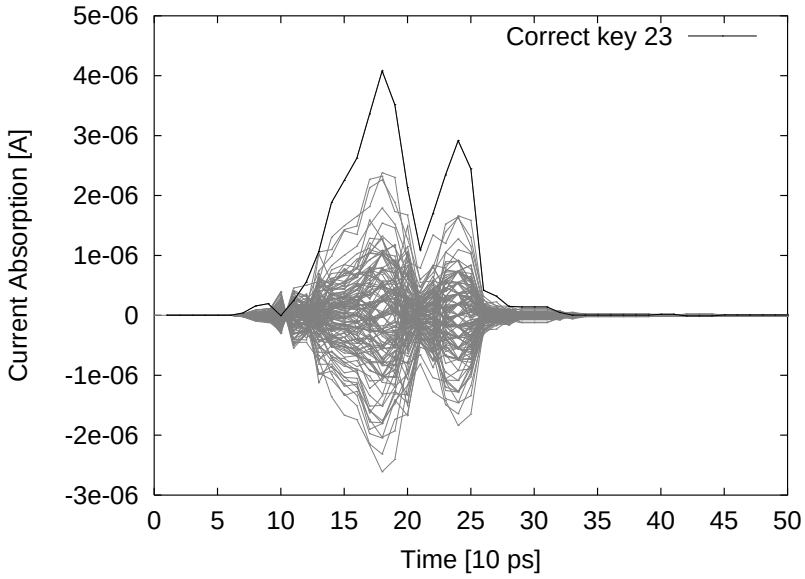


Figure 3.3. Power traces of DPA attack on Kasumi S-box [1]. (©2007IEEE)

will make the attack much more efficient, allowing to isolate the region around which the measurement should be performed. Moreover, depackaging the chip will avoid perturbations due to the passivation layers.

Fault induction attacks exploit some type of variation of external and/or environmental parameters to induce faulty behavior in the components in order to interrupt the normal functioning of the system or to perform privacy or precursor attacks. Faulty computations are sometimes the easiest way to discover the secret key used within the device. Results of erroneous operations and behavior can constitute the leak information related to the secret parameter to be retrieved. Faults can be induced by acting on the device's environment and creating abnormal conditions. Typical fault induction attacks may involve variation of voltage supply, clock frequency, operating temperature and environmental radiations and light. As an example, see [26], where the use of the Chinese Remainder Theorem to improve performances in execution of RSA is exploited to force a fault-based attack. Differential fault analysis (DFA) has been also introduced to attack DES implementations [23].

Scan based channel attacks exploit access to scan chains to retrieve secret information stored in the device. The concept of scan design was introduced over thirty years ago by Williams and Eichelberger [27] with the basic aim of making the internal state of a finite state machine directly controllable and observable. To this end, all (D-type) flip-flops in the FSM are substituted by master-slave devices provided with a multiplexer on the data input and - when the FSM is set to test mode - they are connected in a "scan path", i.e., a shift register accessible from external pins. This concept has been extended for general, complex chips (and boards) through the JTAG standard

(IEEE 1149.1) that allows various internal modes for the system and makes its internal operation accessible to external commands and observation - when in test mode - through the test port. JTAG compliance is by now a universal standard, given the complexity of testing Systems-on-Chip; internal scan chains are connected to the JTAG interface during the packaging of the chip, in order to provide on-chip debug capability. To prevent access after the test phase, a protection bit is set - using for instance fuses or anti-fuses - or the scan chain is left unconnected. However, both techniques can be compromised allowing the attacker to access the information stored in the scan chain [28].

Attacks exploiting NoC implementations

The above attacks were addressed basically to any type of complex architecture. We shall now focus on attacks that exploit the specific characteristics of the NoC architecture. A security-aware design of communication architectures is becoming a necessity in the context of the overall embedded device. While the advantages brought by the use of a communication centric approach appear clear, an exhaustive evaluation of the possible weaknesses that in particular may affect an NoC-based system is still an ongoing topic. The increased complexity of this type of system can provide attackers with new means of inducing security pitfalls, by exploiting the specific implementation and characteristics of the communication subsystem. In addition to the attacks discussed in the previous paragraph, several types of attack scenarios can be identified which exploit Networks-on-Chips characteristics and that derive from networking rather than from chip-based attacks [29, 18, 17].

Denial-of-Service

A Denial-of-Service attack (DoS attack) is an attempt to make the target device unavailable to its intended users. Such attacks may address the overall system or some individual component, such as the communication subsystem. Aim of the attacker is to reduce the system's performances and efficiency, up to its complete stop. This type of attack reaches particular relevance in embedded systems, where reduction of the already limited amount of available resources can constitute a not negligible problem for the device and the users. Effects of a DoS attack to an NoC based system can appear as slowing down of network transmissions, unavailability of network and/or processing and storage cores and disruptions in the inter-core communication. Moreover, the reduced capabilities of the communication infrastructure may compromise real-time behaviors of the system.

We consider hereafter attacks impairing bandwidth (and therefore network resources) and power availability.

Bandwidth reduction attacks aim at reducing network resources available to communicating IPs, in order to cause higher latency in on-chip transmission and conse-

quent missing of deadlines in the system behavior. Depending on the routing strategies adopted, different attacks scenarios can be identified [29]:

- *Incorrect path*. Packets with erroneous paths or invalid origin and destination information are injected into the network, with the aim of routing them to a dead end and occupying transmission channels and network resources, therefore made unavailable to other valid packets.

- *Deadlock*. Packets with routing information capable of causing deadlock with respect to the routing technique adopted are introduced into the network. Packets do not reach their destination, being blocked at some intermediate resource, which in turn, as a consequence, is not available for other transmissions. NoCs implementing wormhole switching are the most likely to suffer from this type of attack.

- *Livelock*. As well as deadlock, livelock is a special case of resource starvation. Packets do not reach their destinations because they enter cyclic paths.

- *Flood (Bandwidth Consumption)*. Aiming at saturating the network, this type of attack is performed by injecting in the network a large number of packets or network requests, such as broadcasting or synchronization messages.

Network interfaces provide a basic filter to requests and packets injected maliciously in the network by compromised cores. However, an illegal access to network interfaces' configuration registers performed by an attacker may be exploited to carry out the described types of attacks. Moreover, fault induction techniques can be applied to modify information stored in such registers and cause disruptions in inter-core communication.

Data and instructions tampering represents a serious threat for the system. Unauthorized access to data and instructions in memory can compromise the execution of programs running on the system, causing it to crash or to behave in an unpredictable way. Therefore, protection of critical data represents a critical task, in particular in multiprocessor Systems-on-Chip, where blocks of memory are often shared among several processing units. Tampering of data and instructions in memory can be performed when a processor writes outside the bounds of the allocated memory, such as for instance in the case of an attack exploiting buffer overflow techniques [21].

Draining attacks aim at reducing the operative life of a battery-powered embedded system. In fact, battery in mobile pervasive devices represents a point of vulnerability that must be protected. If an attacker is able to drain a device's battery, for example, by having it execute energy-hungry tasks, the device will become not more available for the user. Literature [30, 14] presents three main methods for an attacker to drain the battery of a device:

- *Service request power attacks*. In this scenario, repeated requests are made to the victim of the attack. In our context, the victim can be the interconnection subsystem or one or more processing or storage cores. Requests that could be made and that would address the communication infrastructure may involve the establishment of connections to valid or invalid IP cores or the range of memory addresses, as well as synchronization and broadcasting of generic messages. An example of service request power attacks to processing cores is given by the repeated sending of requests to the power manager of the core to keep it in the *Active* state [17, 31].

- *Benign power attacks*. In this kind of attack, valid but energy-hungry tasks are forced to be executed indefinitely. Ideally invisible to the users, these tasks secretly drain the energy source. The attacker provides valid data to a program or a task in order to make it execute continuously and consume a considerable amount of power.

- *Malignant power attacks*. These attacks are mainly based on viruses, worms or trojan horses maliciously installed in the device. The attack alters the operating system kernel or the application binary code in such a way that the execution consumes a higher amount of energy. Malignant power attacks can be for instance performed by a compromised core sending continuous requests to the Bluetooth module. The core will keep the module continuously executing the scan of the available devices and sending requests of connection or malicious files [22].

Illegal access to sensitive information

This type of attack aims at reading sensitive data, critical instructions or information kept in configuration registers on unauthorized targets. Attacks carried out using several agents can be included under this classification. Buffer overflow, described in detail in subsection 3.1.2, can be exploited to compromise a core and use its memory access rights to access an unauthorized range addresses where sensitive data, such as cryptographic keys, are stored [19]. Moreover, side channel information leaking from the device can be detected and used to retrieve secret data or pieces of code.

Illegal configuration of system resources

In this type of attack the aim is to alter the execution or configuration of the system in order to make it perform tasks set by the attacker in addition to its normal duties. The attacks can be performed as a write access in secure areas in order to modify the behavior or configuration of the system. The attacker takes control of one or more resources of the device, and exploits it to achieve its malicious goal. A significant

```
#include <stdio.h>
#include <string.h>

void func(char  p)
{
        char stack_temp[20];
        strcpy(stack_temp, p);
        printf(stack_temp);
}

int main(int argc, char  argv[])
{
        func("This_text_causes_an_overflow!");
        return 0;
}
```

Figure 3.4. Example of an application causing buffer overflow.

example, exploiting buffer overflow in order to reconfigure the setting of peripheral interfaces, is described in [19] for an audio CODEC adopting the IEEE 1394 interface [32]. In the application presented, the CODEC is reconfigured to send unencrypted audio samples to external unauthorized users, in order to bypass Digital Right Management's (DRM) protection.

3.1.2 Protection of critical data

Protection of critical data represents a challenging task in multiprocessor Systems-on-Chip, where blocks of memory are often shared among several IPs. Unauthorized access to data and instructions in memory can compromise the execution of programs running on the systems - by tampering with the information stored in selected areas - or cause the acquisition of critical information by external entities, such as in the case of systems dealing with the exchange and management of cryptographic keys [19].

Attacks exploiting buffer overflow aim at writing outside the bounds of a block of the allocated memory, in order to corrupt data, crash the program, or cause the execution of malicious code. The buffer overflow [21] is probably the best-known type of attack aimed at obtaining illegal memory accesses. In a traditional attack exploiting buffer overflow, the attacker passes as program arguments data whose dimension will exceed the one allocated in the stack buffer. As a result, the information on the stack is overwritten, as well as the return address. The data passed to the program are crafted in order to set the value of the return address to point to malicious code contained in the attacker's data. When the function returns, the malicious code is executed.

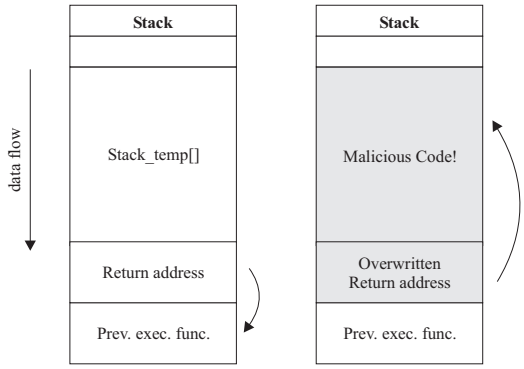


Figure 3.5. Stack behavior during the exploitation of buffer overflow.

An example of a simple application vulnerable to buffer overflow attacks is shown in figure 3.4. In the code shown, *strcpy()* is the vulnerable function. In fact, this C function lacks bounds checking and when the string passed as argument of *func(char *p)* is bigger than the reserved memory space for the buffer, the information passed overwrites the adjacent positions in the stack (see figure 3.5). The input string can be maliciously arranged in order to contain attack code and a return address pointing to the initial instruction of the attack code, which can be for instance a virus or a program aiming at retrieving confidential information from memory. Once the malicious code is running on the compromised core, it can exploit the rights of the application and the core to access the memory. A practical example of exploitation of buffer overflow in embedded systems is presented in detail in [19], where the attacker attempts to obtain a copy of the cryptographic key used to verify the integrity and authenticity of the rights object employed in the Digital Rights Management protocol.

Considering off-chip distributed shared memory multiprocessors [33, 34], the main security problems are caused by the possibility of physically accessing the communication subsystem. In this case, one of the most famous attack is the insertion of a device targeting at tapping or tampering with a bus to change program behavior [35, 33]. For example, Sony PlayStation and Microsoft Xbox can be hacked by *modchips* [35], which can be soldered to the buses on the motherboard, allowing the console to play pirated games. A clean machine always checks the digital signature and media flags from a registered CD by executing a small code in BIOS. A *modchip* monitors the buses to block the code from the BIOS and injects spoofing code at the appropriate time. The hacked machine will execute the new code that skips authorization checks. This kind of attack can be easily performed. In fact, the cost of the *modchip* is around $60, while installing the chip is a relatively easy operation [35].

In multiprocessor System-on-Chip architectures based on an NoC, these direct attacks to the communication subsystem will be unfeasible since there is no physical network access, if not through a debugging or testing interface [28]. The presence of a Network-on-Chip also avoids the possibility of sniffing (or snooping) the information

that is passing through the communication channel, unless the messages are explicitly broadcasted by the initiator of the communication. The only way to replicate the previous attacks in such systems is to use direct read/write operations to the memory subsystem, since it stores the system status.

According to [36], it is possible to classify attacks to memory or to the communication subsystem in three categories:

- *Sabotage*: This first type of attack has as main goal crashing the application or damaging the target system. An example is the possibility to write random values in the memory by overwriting the application source code and its data. This type of attack does not need any knowledge of the target application but, on the other hand, it lacks incentives to be performed since the financial reward for a *sabotage* attack is very limited.

- *Passive*: The main characteristic of this attack is that typically it is non-invasive. In fact, the goal of the attacker is only to steal sensitive information without modifying the system behavior and without being discovered. Reading passwords or economic information directly from the memory are simple examples of *passive* attacks.

- *Active*: This is for sure the most difficult attack to perform since it needs a deep knowledge of the target application but, on the other side, it is the most fruitful for the attacker. *Active* attacks result in an unauthorized state change of the target applications, such as the manipulation of memory data to perform other unauthorized actions.

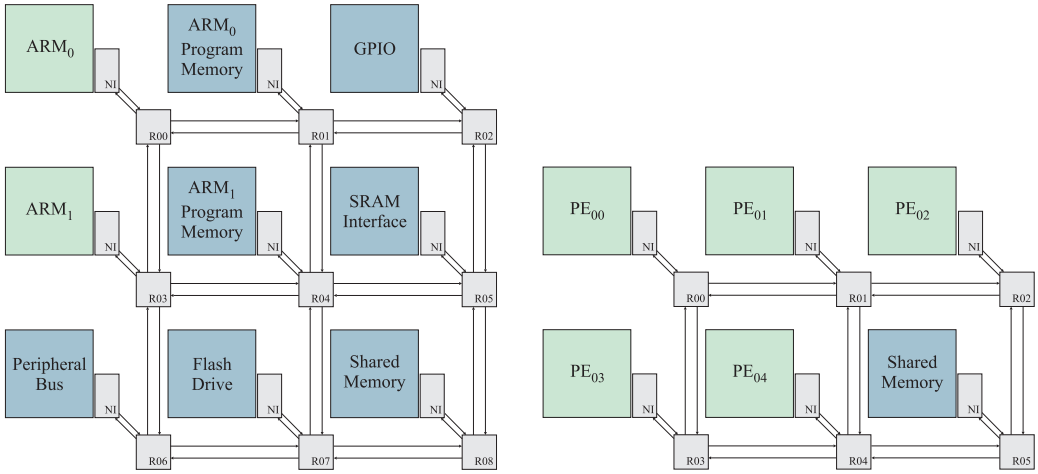
Without making any assumption on the specific capabilities of the attacker to obtain control of processing cores to illegally access the memory, it is possible to notice how, without a careful hardware design, simple software fallacies (such as the buffer overflow) can give the attacker the possibility to perform all the above mentioned memory attacks without much difficulty.

3.1.3 Monitoring system activities to prevent Denial-of-Service attacks

A compromised core executing malicious code can be used to perform a *Denial-of-Service* (DoS) attacks against the system, with the aim of reducing system performance and operative life of battery and device [37]. In DoS attacks, an attacker attempts to prevent legitimate users from accessing information or services. In common data networks, an obvious type of DoS attack can occur when an attacker "floods" a network with information. Since servers are able to process only a certain number of requests at once, in the case of an overload of requests the server can delay processing a legitimate request, causing disruption in the service provided. Similarly, DoS attacks to complex System-on-Chip architectures will target shared resources of the system, such as the communication subsystem or shared memories or peripherals.

While DoS attacks have been studied for Chip Multiprocessors (CMP) [38] and Simultaneous Multithreading (SMT) processors [39], no study has been presented up to now concerning the consequences that DoS attack may cause on architectures adopting the NoC paradigm as interconnecting subsystem. In [38], authors discuss DoS attacks on Chip Multiprocessors (CMP). In such systems, sharing resources such as the L2 cache provides the advantage of reduced inter-core communication overhead and therefore a higher overall throughput for the entire system. However, it also makes the system more susceptible to DoS attacks targeting these shared resources, increasing the possibility of having degradation of performance. Authors design several types of Denial-of-Service attacks and analyze their impact to the performance of CMPs, evaluating how a cracker can exploit the shared resources of a CMP by injecting malicious threads to deprive resources of legitimate applications and cause performance degradation. Microarchitectural Denial-of-Service in SMT processors is discussed in [39]. Microarchitectural DoS attacks occur because a shared resource is exploited in an unexpected manner by one (or more) malicious thread influencing the performance of other threads. For instance, shared resources targeted by the malicious thread may include the instruction cache. If the instruction cache consistency is maintained by flushing the full trace cache, a single action by one thread can impact all threads sharing that same cache. Each thread can be stalled by tens of cycles as the trace cache is rebuilt; during this time, an attacking thread can simply cause the trace cache to be flushed again. Similarly, SMT processor with shared pipeline can suffer from the same type of attack [39]. A Denial-of-Service attack on SMT architectures, based on power density is presented in [40]. In high-performance microprocessors, power density relates to the problem of high power dissipation in a small area causing local hot spots in the chip. In the DoS attack discussed in the work, a malicious thread repeatedly accesses a resource to create a hot spot at that same resource. If the resource targeted by the attacks is shared, the hot spot affects all the threads and it is harder to identify the source of the problem. Several solutions to identify the occurrence of the attack and to mitigate its effects on the overall system performances are discussed.

As discussed in subsection 3.1.1, in the case of NoCs, DoS attacks such as *Bandwidth consumption* or *Draining attacks* can be easily performed once the malicious code takes over one of the processing elements. In the remaining part of this subsection we present the results of a set of experiments we performed to evaluate the effects of flooding attacks on the two different architectures shown in figure 3.6, i.e., a typical *embedded architecture* composed of 2 processors (initiators), 1 shared memory and 6 generic targets, and a *shared memory multiprocessor* composed of 5 initiators and 1 target shared memory. The experiments consist in simulating the effects of a DoS attack generated by running a simple malicious code on one of the processors of the architectures, and in measuring increases in links utilization and performance loss in the system. As processing elements, we consider processors of the ARM9 family [41], with data and instruction cache memories of 8 KBytes (for instance the ARM922T).



(a) Embedded architecture with 2 initiators, 1 shared memory and 6 generic targets.

(b) Multiprocessor architecture composed of 5 initiators and 1 target memory.

Figure 3.6. Architectures considered in the experiments implementing the DoS attack.

One memory block is shared among the processing elements acting as initiators. The NoC is a mesh, implementing a transaction-based protocol, and a XY routing. In the experiments, length of buffers in routers is of 4 slots, while buffers in NIs are 16 slots long and able to store completely one packet. The NoC provides the cores with a Quality-of-Service class of type *Best Effort* (BE) [9], and the switch allocator of the routers assigns the use of output ports by employing a Round Robin algorithm. More advanced techniques exist to guarantee a given throughput to each communication in the NoC [9]. While these techniques may mitigate the effects of the DoS attack on the communication infrastructure, they do not modify the effects on the shared resource utilization (e.g. the shared memory). Moreover, depending on the implementation of the Quality-of-Service techniques of the NoC, the compromised core can be allowed to reserve connections for a long time (for instance using *lock-down* operations), potentially amplifying the effects of the attack on shared resources.

Fig. 3.7 shows the simple malicious code we designed to cause the DoS attack in the systems presented in figure 3.6. The code is crafted with the goal of reducing the bandwidth available to the various initiators, as well as decreasing the availability of the shared memory, by generating frequent cache misses. Frequent data cache misses are generated due to the fact that in *C* matrices are stored by rows (row-major order), while in the code presented *array* is accessed by columns instead that by rows. Without loss of generality, we assume in our experiments the malicious code running on a core different than the processing elements executing the main application. Alternatively, the malicious code could be run as thread of the main processor.

```
define size 100000000
#define limit 10000

int array[size];

int main()
{
        int i = 0;
        int j = 0;

        for (j = 0; j++; j < size)
        {
                array[i] = j;
                i+=limit;
                if (i >= size) i = 2;
        }
}
```

Figure 3.7. Code used to perform the DoS attack.

In our experiments, we focus on the traffic generated by the communication transactions among initiators and the block of shared memory. The malicious code is able to increase the time needed to complete a memory transaction and therefore to degrade the performance of the system.

We simulated the attack by implementing a SystemC NoC simulator. We instrumented the code in order to extract information about the time needed for memory transactions and the links utilization. Five multimedia applications were considered in the experiments: four from the MiBench embedded benchmark suite [42], and the H263 video codec. We traced memory requests and cache misses generated by the applications running on an ARM processor model simulated with the SimpleScalar/ARM simulator [43]. Cache misses generate load or store requests to the shared memory, that are converted by the NI in NoC in packets that are forwarded by the NoC to the NI of the target memory. Once the memory operation is completed, data requested (or an acknowledgement signal) are sent back to the initiator of the transaction. The storage element was modeled as a single write/read port memory. In our simulations, we neglect traffic due to system synchronization and to other internal communications, focusing only on the effects of the DoS attack on the performance degradation of the cores running the benchmark. Therefore, the results we obtain are optimistic. In fact, under this assumption, we reduce the global traffic on the NoC, as well as the number of possible conflicts for using shared resources. We run the NoC simulator for 10'000'000 clock cycles, skipping the first 100'000 cycles to avoid the transients due to the initialization of caches and NoC shared resources. In the first architecture

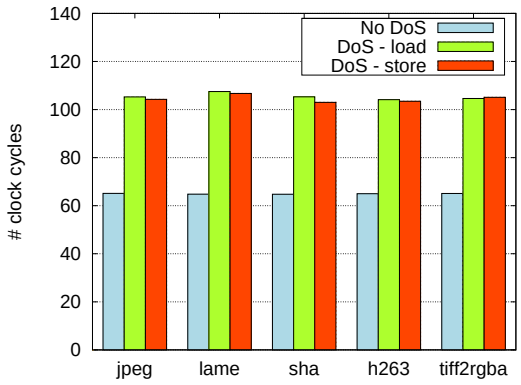


Figure 3.8. Transaction latency for several benchmarks in the embedded architecture.

Table 3.1. Performance loss due to the DoS attack.

Benchmark	Miss rate	CPI reduction (%)
jpeg	0.0105	40.29
lame	0.0180	42.59
sha	0.0007	40.34
H263	0.0064	40.13
tiff2rgba	0.0082	39.27

considered (*embedded architecture* - figure 3.6(a)), ARM_0 runs the benchmark, while ARM_1 runs the malicious code. In the second architecture (*shared memory multiprocessor* - figure 3.6(b)), we perform several experiments by changing the core running the malicious code, while letting the remaining cores run copies of the benchmark.

Effects of the DoS attack on the embedded architecture

Figure 3.8 shows the average latency, in terms of clock cycles, of a transaction from the ARM processor running the benchmarks. Values are presented for the case in which the core is not yet compromised and the one in which the system is under the effects of the DoS attack. Results show the effects of attacks in the case of *load* and *store* transactions. While for all the benchmarks the transaction latency is almost the same in the case of normal behavior, the effect of the DoS attack is more relevant when executing those benchmarks requiring a higher number of accesses to the memory, i.e., those that present a higher number of cache misses. The *lame* application results to be the most sensitive to the DoS attack, with an increase of the transaction latency of around 66%. As it can be noticed from figure 3.8, the effects on the transaction latency of the DoS attack carried out with *store* operations does not differ significantly from those obtained with the *load* operation.

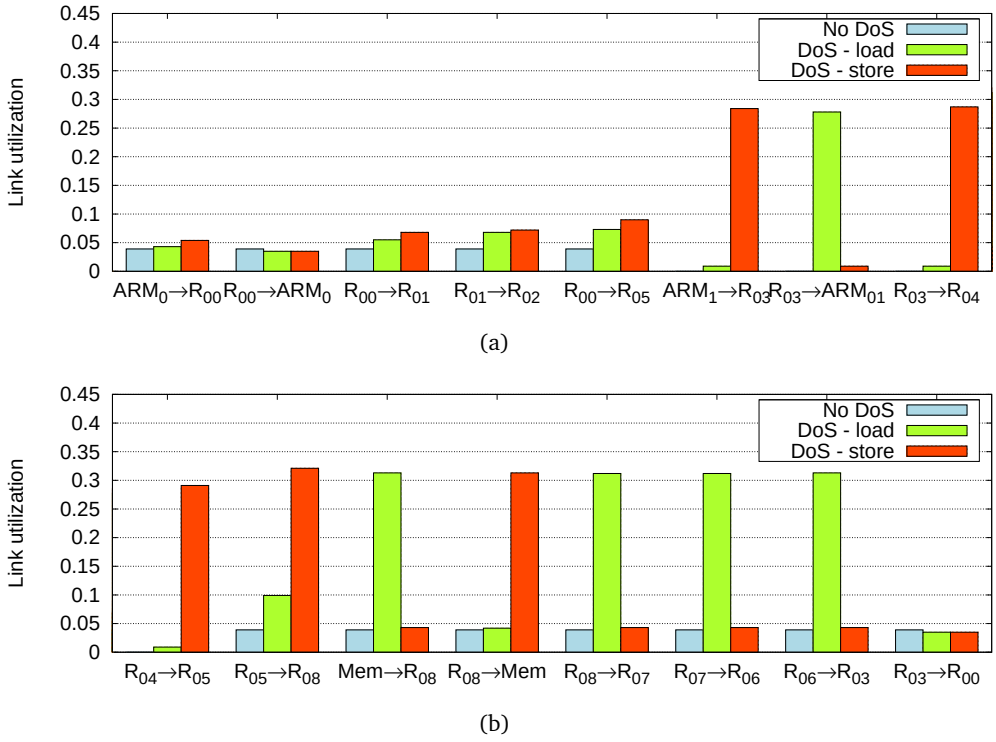


Figure 3.9. Link utilization for the embedded architecture.

Table 3.1 summarizes the measured performance loss of the main processor for the several benchmarks. The reference average CPI of the ARM9 family is 1.76 [41]. The bigger loss can be seen in the execution of the *lame* benchmark, where the CPI reduction is up to 42.59%. For all the benchmarks, the reduction is above the 39%. This result was again expected, since the *lame* benchmark is the one that access more frequently the remote shared memory.

Figure 3.9 shows link utilization in the NoC, in the case of the *lame* benchmark. Link utilization is measured as fraction of the use (value 1 is equivalent to a link always occupied transferring data). The figure shows only the links crossed by the packets following the XY routing strategy. In the figure, the notation $a \rightarrow b$ indicates the NoC link from element a to element b, where a and b are the NoC components as called in figure 3.6(a). It is possible to notice how the influence of the DoS attack on the link utilization depends on the relative position in the NoC of the compromised core. In the case of the *load* instructions, the links more affected by the attack are those on the path taken by the *ack* signals from the memory to the compromised core. In the case of *store*, links more affected are situated on the path from the compromised core to the memory. This is due to the fact that a *load* instruction issued by the compromised core implies the transfer from the memory to the core of a relevant amount of data that

tend to occupy the NoC resources on that path, while for the *store* instruction the data are mainly transferred from the compromised core to the memory.

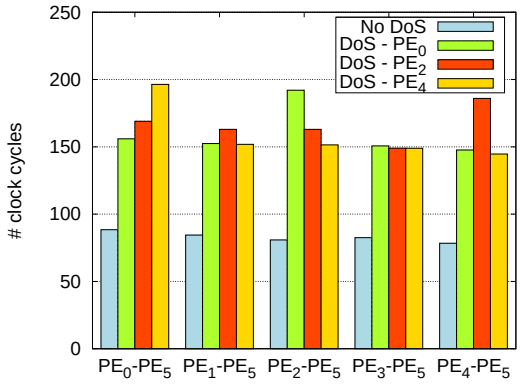
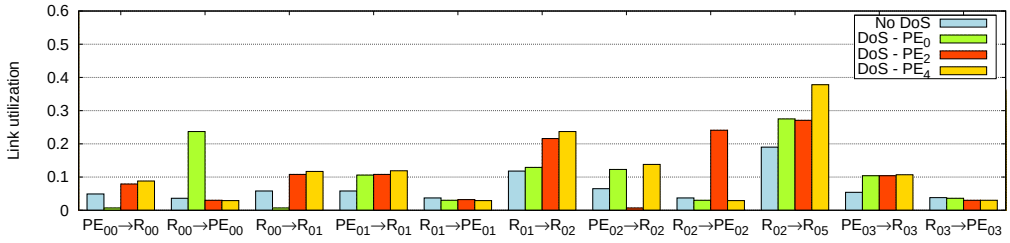
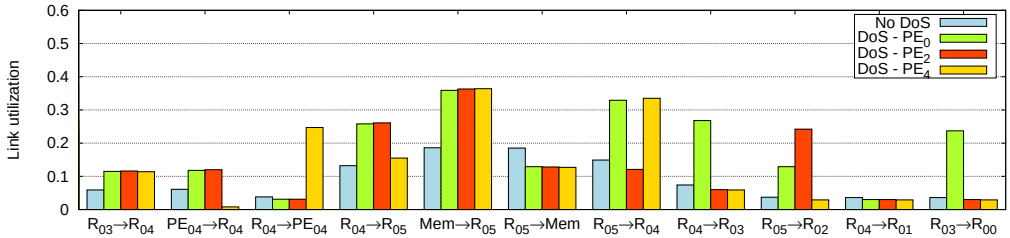


Figure 3.10. Transaction latency in the multiprocessor architecture.



(a)



(b)

Figure 3.11. Link utilization for the multiprocessor architecture.

Effects of the DoS attack on the shared memory multiprocessor architecture

Figure 3.10 shows the average latency, in terms of clock cycles, of a transaction from the processing cores running the *lame* benchmark. Results are shown for the case in which the malicious code is executed in PE_0, PE_2, and in PE_4. The DoS attack causes

an increase of the transaction latency to up the 91%, therefore notably influencing the overall system performance.

Figure 3.11 shows links utilization for the NoC, in the two cases analyzed. It is possible to notice how the usage of the links, in particular of the one involved in the communication generated by the malicious code, dramatically increases. Also in these cases, the increase in the links utilization depends on the position of the compromised core in the NoC.

3.2 Monitoring NoCs

The integration of large numbers of heterogeneous processing cores, storage elements and I/O peripherals in the same platform poses additional challenges to application designers. A high number of complex concurrent applications will share the available resources providing users with new services and functionalities, while platform-based design will allow to reduce the cost per single item by giving the system the possibility to easily adapt to different application requirements. The efficient exploitation of available resources in these complex systems represents however a challenge for application designers, and new tools are needed for NoC platforms, which employ information derived by a measurements taken on the running system by an ad-hoc implemented monitoring system.

On the one hand, in order to reduce the cost per single item and the time-to-market, hardware platforms are in general over-designed in terms of the offered functionalities. By following the platform-based design approach, the same platform is customized for specific applications by turning off unneeded resources, giving the system the possibility to easily and rapidly adapt to different application requirements and/or customer needs. The complexity of these systems raises the problem of efficiently exploiting the amount of resources available, and of understanding the system behavior once the platform has been implemented. A monitoring system is therefore needed for helping designers in the tasks of optimizing resources' utilization, by exploiting information derived from measurements taken on the running system. Modern, high performance processors deal with this problem through the use of special on-chip hardware that can monitor performances and system behavior [44]. By using a set of selected hardware event detectors and counters, information concerning system elements is collected during program execution, analyzed, and then exploited to support the optimization of users' applications and the performance improvement. In the case of NoC architectures, the presence of a highly distributed environment poses several challenges, in terms of event detection and data collection, that should be addressed by the platform designers.

On the other hand, the large amount of hardware resources present makes it possible to execute on the same device a very large number of diversified multimedia and mobile applications. The execution of these applications introduce the necessity to

match their requirements with the resources of the platform. However, applications' requirements are usually not fully known at design time, in particular in the case of changing run-time conditions or in the case of multiple tasks executing with different computational effort, communication bandwidth, power consumption, and competing for the same shared resources [45]. In order to cope with uncertainty, nowadays systems are over-sized, and their resources are not optimally used with a static and separated allocation for each critical service. In future devices, it is foreseen that adaptivity of systems will ensure optimal dimensioning and utilization of resources as well as enhanced predictability via a dynamic and global run-time management [45, 46, 47]. Having the possibility of monitoring run-time activities and reacting to unpredicted changes of the system behavior enables to deal with the problem of designing complex system whose behavior could be only in part foreseen at design and compile time by system designers.

For both types of design aspects, an efficient and reliable monitoring system is needed for measuring resources utilization and monitoring system behavior. Similarly, measurements performed on an instance of the application running on a *virtual platform* [48, 49] can be used for the profiling and optimization phase at design time. *Virtual platforms* simulate the behavior of their target architectures, and by specific instrumentation of the simulator code it is possible to measure the desired application characteristics at execution time. However, in the case of application scenarios changing at run-time, a monitoring system embedded into the actual platform is needed to support the detection of modification in the application characteristics and therefore the run-time adaptivity of the system. Being the NoC the central element of architectures based on the communication-centric paradigm, it represents the ideal means for collecting the information about the individual cores as well as about the system as a whole.

In this dissertation, we will address the monitoring of NoC activities, in particular focusing on the events that can be detected from the network interface. For this type of monitoring, we identified four main categories of events to be taken into account: *throughput characterization*, *timing/latency* proprieties, *resource utilization (buffers)*, *events and messages statistics*.

- **Throughput characterization** deals with the measurement of the amount of data transmitted and received by the cores. Data collected can provide information useful for optimizing the distribution of application tasks in the system, for verifying that the bandwidth available to the core is sufficient for guaranteeing its communication requirements, or for optimizing at run-time the power consumption due to data transmission. An example of possible applications that can benefit from monitoring the run-time throughput generated by processing cores is given in [50]. By measuring the amount of traffic dynamically generated by the cores, it is possible to adapt the traffic parameters to meet the performance requirements needed in those SoC platforms implementing mobile, multimedia

and 3DTV/HDTV applications for which, becoming more general purpose than in the past, it is no longer possible to clearly identify a set of reference use cases.

- Information about **timing** and **latency** plays a significant role in systems in which latency in communication is critical (for instance for memory-intensive applications), as well as for the estimation of the length of tasks execution in target cores, both for helping the operating system scheduling, and for identifying possible bottlenecks. As shown in [51], this is particularly important in systems such as Mobile Internet Devices, which can potentially execute at the same time communication services, multimedia playback, content creation and office applications, with the need of simultaneously providing the applications with guaranteed throughput and best-effort services, while limiting communication latency for traffic generated from general-purpose applications running on processors with caches, components system performance often dramatically depends on [52].

- For motivations similar to those aforementioned, **resources' utilization** can be monitored for optimizing bandwidth allocation, as well as for discovering possible run-time problems of the communication system, such as the violation of Quality of Service (QoS) contracts between cores and NoC. In systems like the one presented in [53], the Operating System (OS) monitors NoC communications by polling the network interface through a remote function call, in order to obtain the traffic statistics and the utilization of the NI's queues. By exploiting this information, the OS is able to control the message injection rate by limiting the time wherein a certain processor is allowed to send messages onto the network, as well as changing routers' routing table for diverting message streams from one channel to another.

- Detection of **NoC events' and messages' characteristics**, among the others, is useful to verify the behavior of network control messages, as well as to tune network characteristics (messages length and patterns, message injection rate, etc) to better satisfy application requirements, in particular in adaptive systems. Information detected can be employed for debugging purposes, as well as for helping tracing behaviors and changes on the system configuration at run-time. Monitoring this type of events can also be useful for security purposes [18, 54]. Events such as the overflow of the expected message length in FIFOs or the tentative of accessing restricted memory addresses are associated with potential malicious behaviors of processing elements, and lead to possible isolation of cores or network reconfiguration.

In addition to the list of detectable events, another aspect to be considered in the design of a monitoring architecture is its influence on the observed system, i.e., the so called *probe effect* [55]. It refers to the fact that any attempt to observe the

behavior of a generic distributed system may change the behavior of the system itself. In general, the monitoring system should be able to minimize or completely avoid to influence with its operations the measurements extracted on the system's timing and execution properties, in order to avoid non-deterministic behavior in programs with race conditions and poor synchronization [56, 57, 58]. Besides the detection of the events, the characteristic of non-intrusiveness concerns also the collection and storage of monitoring data, that should not influence the nominal data transmission. Moreover, it is possible to distinguish between the *passive* and *active* monitoring of a system, the former using devices that just observe the occurrences of the events and the latter also injecting test signals for observing the results of the injection on the behavior of the system. In our work, we consider *passive* monitoring, therefore focusing on those events that can be observed by the network interface, both in terms of applications behavior and in terms of the communication system's behavior.

3.3 Fault susceptibility of NoCs

As complexity of design increases, and as CMOS technology scales down into the deep-submicron domain, devices and interconnect are subject to new types of malfunctions and failures that are hard to predict and to avoid with the current design methodologies [59, 60]. This is particularly true for embedded systems, often composed of a high number of heterogeneous IP cores (possibly offered by different vendors), and connected by means of a Network-on-Chip (NoC). In order to deal with faults in such complex systems, new fault tolerant approaches are needed: new methodologies and architectural solutions should be explored.

The network interface represents a critical point in the design of a fault tolerant NoC. NIs are in charge of interfacing IP cores to the communication infrastructure, and therefore, to the overall system. Only few works have addressed up to now its fault tolerance. Faults in the NI can cause errors that, directly affecting the correct transmission of data and control information, could be extremely hard to detect and recover without the appropriate support (leading for instance to *deadlock* or *livelock* conditions). Moreover, a faulty NI can isolate a working core (or cluster of cores) from the rest of the system, thus generating a massive and unwanted extension of the fault area.

Particular attention should therefore be given to the design of the fault tolerant provisions of the NI. Usual fault tolerant hardware implementations of sensitive components employ triple module redundancy (TMR), in which three copies of the same component perform the same operation, and the single output result is obtained by a voting system [61]. A TMR implementation is however expensive in terms of the amount of resources and energy needed; in particular in the case of embedded systems, the strict design constraints make the extensive use of hardware redundancy not economically viable. This is particularly true for NIs, which often represent a signifi-

Table 3.2. Fault injection results.

Component	Fault location	Fault percentage (%)
NIs	LUT	18.41
	Buffers	30.05
	FSMs	0.21
	Other	2.67
	Total	51.34
Routers	Input buffers	30.53
	Switch allocators	3.33
	FSMs	0.42
	Other	14.38
	Total	48.66

cant part of the area of the overall NoC [62].

In this dissertation, we focus on providing protection to the NI from both soft and hard errors.

Soft errors, also called single-event upsets (SEUs), are caused by the interaction of the system with radiations such as neutrons from cosmic rays and alpha particles from packaging material. While traditionally this type of concerns regarded mainly space applications, with newer technologies soft errors are much more frequent than in the previous generations [63]. Moreover, embedded systems often operate in harsh environments in which the soft error rate (SER) is significantly higher than in normal operating conditions at ground level. The SER increases also in devices implementing dynamic frequency and voltage scaling (DFVS) techniques for energy management [64], as well as varying often unpredictably with technology scaling, process variation, and speed operations. The number of soft errors affecting the device is directly proportional to the amount and dimension of storage elements in the system. Typical values for the SER, reported for modern SRAM cells, are at ground level in the range of 1000 to 10000 Failures in Time for Mbit of storage components (FIT/Mbit), while more than 15 times those values at 60'000 feet [65]. SER of the same order of magnitude can be also found for flip-flops implemented in newer technologies [63].

In this work, we also consider permanent faults. These may derive from defects occurred during the manufacturing process and that were not correctly identified with standard testing techniques at the end of production, as well as from defects that occur during the system's lifetime. Considering this second case, for example, the interaction with high energy particles can permanently damage memory cells (such faults are estimated to be around the 2% of the total soft errors [65]); wear-out mechanisms such as electro-migration, gate oxide breakdown, hot carrier injection, and negative bias temperature instability can moreover cause permanent failures in the device [66]. Hard fault causes are extremely varied and may be process-dependent; as a consequence, rather than pinpointing the specific cause, in this dissertation we will adopt higher-level fault models, as done, in general, in the literature on fault-tolerance.

In order to understand the susceptibility of the network interface and the NoC, we employed a fault injection system. As model for the NI, we consider the baseline NI architecture shown in figure 2.1, while for the routers we implemented a version of the input buffer architecture presented in chapter 2. In the experiments, we considered a tile-based NoC in a mesh topology. The NoC implements a wormhole control flow and a deterministic deadlock-free source-based routing. Without loss of generality, in these experiments we implement an XY routing. Network packets are 10 flits long. The header is contained in the first flit of the packet. We considered a 34-bit data-path (32 bits for data and 2 bits for control information) and a depth of 4 for the input buffers of routers and 8 for input and output buffers of NIs. Both network interfaces and switches were implemented in VHDL, and synthesized with Synopsys Design Compiler, by employing the Nangate 45nm CSS typical open cell technology library [67]. In our synthesis, we targeted a frequency of 500 MHz. In the case of a 4x4 tiled mesh topology, the area obtained for the NI and the router is respectively $0.0077mm^2$ and $0.0108mm^2$, while the number of flip-flops for the two components is 965 and 1140, respectively.

We use a random traffic generator system to exercise the NoC components during the fault injection campaign. Every node generates packets random destination nodes with an injection rate equal to 0.01 flits per cycle per node. The fault injection simulation was run for 10 Mcycles. The system evaluated a total of 43'921 faults.

In the fault injection campaign, we focused on Single Event Upsets. SEUs were modeled as bit flips that occur in any of the NIs and routers storage cells (registers, flip-flops, and latches). Bit flips are assumed to occur at random times and location during the circuits' operation. We chose to consider at this level only SEUs because errors induced by such faults in our reference architecture (and the detection/recovery countermeasures we introduce) provide a good representation of the ones induced by hard faults as well. While this is obvious for hard faults in the memory cells and in registers, other faults will also lead to corrupting the information derived from the NI's functions as detailed here below; since our approach targets errors and operates at functional level, checking SEUs coverage implicitly checks for coverage of the hard faults as well. The faults non exercised by our fault injection campaign involve mainly glue logic, that represents less than 4% of the total NI's area.

Table 3.2 reports the percentage of faults measured for each component of NIs and routers. Both in the case of the NI and the router, *Other* components include glue logic and some additional registers needed for the control of the FIFOs. For a 4x4 mesh topology, the number of faults concerning NIs and routers are approximately similar.

By analyzing the functional architecture, we identified the following types of functional errors:

1. **Corrupt Data Error:** data are corrupted during the operations of the NI and wrong data are sent through the communication channel. This type of error can happen due to faults in the protocol adapters and in the FIFOs;

Table 3.3. Errors measured in NIs during the fault injection.

	Error Type									
	Corr. data		Corr. prot. conv.		Routing path		Control flow		No effect	
Fault location	No.	%	No.	%	No.	%	No.	%	No.	%
LUT	0	0	0	0	154'040	81.07	0	0	135	0.07
Buffers	16	0.01	443	0.23	0	0	0	0	12'739	6.07
FSMs	0	0	92	0.05	0	0	0	0	0	0
Other	3	0.001	390	0.21	0	0	48	0.03	731	0.38
Total	19	0.011	925	0.49	154'040	81.07	48	0.03	13'605	6.52

Table 3.4. Errors measured in routers during the fault injection.

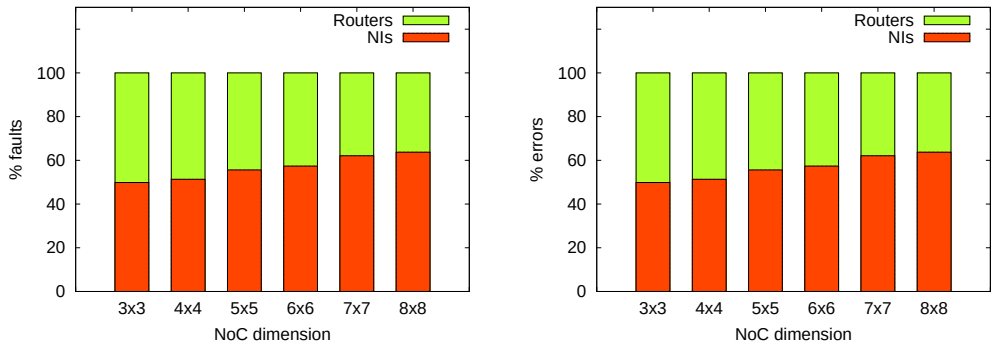
	Error Type											
	Corr. data		Routing path		Mult. copies		Control flow		Prot. error		No effect	
Fault location	No.	%	No.	%	No.	%	No.	%	No.	%	No.	%
Input buffers	11	0.006	3	0.002	0	0	2	0.001	610	0.321	12'785	6.729
Switch allocators	0	0	0	0	490	0.205	0	0	98	0.051	975	0.513
FSMs	0	0	0	0	0	0	0	0	182	0.096	0	0
Other	29	0.015	8	0.004	0	0	379	0.199	0	0	5'901	3.11
Total	40	0.021	11	0.006	490	0.205	381	0.2	890	0.468	19'661	10.352

2. **Corrupt Protocol Conversion Error:** on the side of the node initiating the transaction, faults in the NI lead to control signals received from the core being corrupted, causing the NI kernel to generate wrong routing and control information for the packet header. On the target node side, a fault affecting the protocol conversion will cause a wrong implementation of the core communication protocol, invalidating or disrupting the operation performed. This type of error is due to faults in the NI's protocol adapters;

3. **Routing Path Error:** routing paths inserted in packets' headers are calculated looking up the addresses of requested operations. Faults in the lookup table (LUT) cause erroneous routing and control information to be inserted in the packet headers, leading to possible communication errors, such as misdirection, deadlock, or livelock. Similarly, faults in the FIFOs, when storing this information, can cause this type of error;

4. **Control Flow Error:** faults in registers storing control information in FIFOs and protocol adapters cause errors in the control flow of the FIFOs, by communicating corrupted information about the flits in the buffers. For instance, multiple copies of an outgoing or incoming packet could be sent to the input or output port throughout the time.

Table 3.3 shows errors generated on the NIs by the faults injected, according to the high-level error model just described. The table shows the measured number of errors and the related percentage with respect to the total errors in the NoC. As it is



(a) Faults distribution

(b) Errors distribution

Figure 3.12. Total faults and errors involving NIs and routers when varying the number of nodes in the NoC.

possible to note from the table, a significant percentage of the faults leads to *Routing Path Errors* (around the 81%). This is due to the fact that while bit flips in buffers would be overwritten by new inserted data, faults in the LUT will continue to generate packets with erroneous path information until the stored information is updated or reset. The second most significant type of error (*Corrupt Protocol Conversion Error*) is due to faults mainly concerning NIs' buffers' and NIs' control registers, and it is likely to lead the NI to a crash, i.e., to a situation in which the NI reaches a state in which its operations are definitely compromised, and a hardware reset is needed in order to have the system working again correctly. It has to be noticed that the results shown in table 3.3 strongly depend on the traffic distribution, on the packet injection rate, and on the dimension of the NoC. The experiments performed consider a relatively low utilization of the NoC, for which buffers are most of the time empty. Therefore, SEUs hitting buffers have low probability of affecting flits stored in FIFOs, causing no errors in the NI (6.07%).

Table 3.4 shows errors generated into the routers by the faults injected. The table shows the measured number of errors and the related percentage with respect to the total errors in the NoC. In the case of the router, we adapt to our implementation the system level fault models defined in [68] and [69]:

- **Corrupt Data Error:** transported data are corrupted during its passage through the router;
- **Routing Path Error:** the data packet is routed to a direction different than the one implied by the routing information originally inserted in the header of the packet;
- **Multiple Copies Error:** packets are duplicated and sent to multiple ports;

- **Control Flow Error:** the control flow of the FIFOs is corrupted, due to faults in registers storing control information in the router;
- **Protocol Error:** a temporary or permanent fault in the state register of a Finite State Machine implementing the router protocol leads the router to a state from which it cannot recover.

With respect to the total router's faults, a significant percentage of the faults leads to a router crash. Reasons for this are due to faults in the control registers of the buffers, to packets stuck in the router for faults in the switch allocators, and to corrupted FSM state registers. *Multiple Copies Errors* represent another significant type of error for routers, which count for the 0.2% of the total NoC's errors. Similarly to the results obtained for the NIs, the errors measured depends on the traffic characteristics, and on the utilization of the resources when hit by the faults. For the case of the routers, faults not leading to any error count for approximately 10.4%.

As tables 3.3 and 3.4 show, NIs represent the main source of errors in the NoC (around 88.7%). Figure 3.12 shows how the distribution of faults and errors in NIs and in routers changes when varying the number of nodes in the NoC. Experiments were performed by imposing the same traffic condition, topology, and faults generation than in the case of the 4x4 NoC. The increasing influence of the NI in the number of total faults and errors is due to the fact that, while the total area of the routers is approximately directly proportional to the number of nodes n in the topology, the total area of the NIs (and in particular of the LUTs) increases with n^2. This fact causes NIs to have higher probability to be hit by faults when the number of nodes increases, and, consequently, to lead to errors in the system.

3.4 Summary

This chapter described the motivations for choosing *security*, *monitoring*, and *fault tolerance* as high-level services to be implemented in the NoC.

With respect to the aspect of security in NoCs, in subsection 3.1.1 we showed several type of attacks that can target NoC based architectures, and embedded systems in general. We focused in particular on attacks aiming at accessing critical information stored on the system, or at compromising the system's behavior. The secure communication architecture proposed in chapter 5 is conceived to deal with the previously described memory-based attacks and weaknesses intrinsic of a multiprocessors system with shared memory, exploiting the characteristics of the Network-on-Chip to detect and to prevent them.

With respect to the monitoring of NoCs, subsection 3.2 pointed out that a careful design of the monitoring system is needed, which involves the design of probes detecting the measured events, and of methods for collecting, storing, and processing the collected information. Moreover, subsection 3.2 presented a taxonomy of events that

an NoC monitoring system should consider, as well as discussing general characteristics of data collection and storage. The design of an NoC monitoring system will be discussed in chapter 6.

With respect to the aspect of fault tolerance, in subsection 3.3 we showed, by performing a fault injection campaign, that the NI represents a critical element in the design of a fault tolerant NoC. As figure 3.12 shows, this fact is particularly true the number of nodes in the NoC increases. These results calls for the need of designing ad-hoc fault tolerant solutions for NI's components. In chapter 7, we will address in particular the fault tolerant aspects of network interfaces.

Chapter 4

Related Work

Networks-on-Chip have been, since the first papers proposing the subject [6, 7], a scientific topic in which researchers have been particularly active. Even if the NoC idea seems an adaptation to the SoC context of distribute computers network concepts, many research issues are still open, due to the different constraints and amount of resources available in the NoC case. Key open points in current literature are, for instance, choice of network topology [70, 71, 72, 73, 74, 75, 76, 77], routing strategies [78, 79, 80, 81, 82, 83, 84, 85, 86], flow control [78, 79, 87, 88, 89], queuing sizing and management [78, 90, 91, 92], and methods to guarantee Quality of Service (QoS) at low hardware cost [93, 94, 95, 96, 97, 98, 51]. Asynchronous implementations of NoCs have been proposed and evaluated by different research groups [99, 100, 101, 102], as well as implementations based on optical and wireless technologies [103, 104, 105, 106, 107, 108], or on 3D chips [109, 110, 111].

In the following sections, we will present and discuss related work about NoCs which focuses on the high-level services considered in this dissertation, namely, security, run-time monitoring, and fault-tolerance.

4.1 Security in Networks-on-Chip

Security is gaining increasing relevance in the design of embedded systems, intrinsically constrained in terms of computing power, area, and energy consumption. While architectures for general purpose and embedded processors have been the object of interesting studies in [112, 113, 12, 11, 19], only recently the aspects of security more specifically related to NoC-based systems have been taken into account. A specific solution to secure the exchange of cryptographic keys within an NoC was presented in [114] and [115]. The work addresses in particular the protection from power/EM attacks of a system containing non-secure cores as well as secure ones, the "secure cores" being defined as hardware IP cores which can execute one or more security applications. The framework supports authentication, encryption, key exchange, new user

keys and public key storage, and other similar procedures. No unencrypted key leaves the cores of the NoC and only secure IP cores running trusted software are supported. At the network level, security is based on symmetric key cryptography, where each secure core has its own security wrapper storing a private network key in non-volatile memory.

Diguet et al. [18] propose a first solution to secure a reconfigurable SoC based on NoC. The system is based on *Secure Network Interfaces* (SNIs) and on a *Secure Configuration Manager* (SCM). The SNIs act as filter for the network and handle attack symptoms that may be caused by denial of service attacks and unauthorized read/write accesses. The SCM configures system resources and network interfaces, monitoring the system for possible attacks. A routing technique based on reversed forward path calculation is proposed in [29]. The technique allows verifying that the sender of the packet arrived at a specific SNI has the right to communicate with it.

It is worth noting that the use of protected transactions is also included in the specifications defined by the Open Core Protocol International Partnership (*OCP-IP*) Association [8]. The standard OCP interface can be extended through a layered profile in order to create a secure domain across the SoC and provide protection against software attacks and against some selective hardware attacks. The secure domain might include CPU, memory, I/O etc... that need to be secured by using a collection of hardware and software features such as secured interrupts, secured memory, or special instructions to access the secure mode of the processor.

Memory data protection in general embedded systems architectures is another subject that should be considered as work related the NoC security solutions presented in this dissertation. A specific implementation of a protection unit for data stored in memory is described in [19]. The proposed module enforces access control rules that specify how a processing element can access a memory device or peripheral in a particular context. AMBA bus transactions are monitored in order to discover specific attacks, such as buffer overflow to steal cryptographic keys used in Digital Right Management. A lookup table, indexed by the concatenation of the master identifier signals and the system address bus, is employed to store and check access rights for the addressed memory location and to stop potential non-allowed initiators.

Considering commercial implementations of on-chip memory protection units, *ARM* provides, in systems adopting the ARM TrustZone technology [116], the possibility of including a specific module - the AXI TrustZone memory adapter - to support secure-aware memory blocks. A single memory cell can be shared between secure and non-secure storage area. Transactions on the bus are monitored to detect addressed memory region and security mode, in order to cancel non-secure accesses to secure regions and accesses outside the maximum address memory size. The module is configured by the TrustZone Protection Controller, which manages the secure mode of the various components of the TrustZone-based system and provides the software interface to set up the security status of the memory areas.

A solution similar to the one presented above is provided by *Sonics* [117] in its SMART Interconnect solutions. An on-chip programmable security "firewall" is employed to protect the system integrity and the media content passed among on-chip processing blocks and various I/Os and the memory subsystem. The firewall is implemented through an optional access protection mechanism to designate protection regions within the address space of specified targets. The mechanism can be dynamic, with protection region sizes and locations that can be programmed at run-time. It can also be role-dependent, with permissions defined as a function not only of which initiator is attempting to access but also which processing role the initiator is playing at that time. Protection regions subdivide a target's address space, where each target can have up to 8 protection regions. One of four levels of priority is assigned to each protection region.

All the above refers to work published before the year 2008; obviously, security in NoC-based systems has been studied also in more recent research work. While it would be unfair to compare our work to research published after it, it is worth mentioning research directions explored by other authors. Protection of sensitive data and implementation of methods for restricting the access to sensitive memory blocks in multiprocessor systems have been the topic of several research articles, often proposing solutions similar to the one proposed in this dissertation. A reconfigurable data protection controller for NoCs is proposed in [118], while the use of hardware modules embedded into the NI for supporting memory access control and additional security services is discussed in [119], [120], and [121]. In particular, in [121] the concept of Quality-of-Security Service is introduced, by adopting different security levels. Each level represents a trade-off between security and performances. Generic security frameworks adopting some of the concepts introduced in the work described in this dissertation have been applied in [122] and [123], in particular involving the use of additional blocks for enforcing access control in memory blocks, and the use of ad-hoc probes for monitoring security-related system behavior.

In [124], security in MPSoCs is enhanced by using an Authenticated Encryption method for dividing the NoC into secure and non-secure zones, while in [125] authors propose to use asynchronous design for reducing power consumption spurs and making NoC communications less affected by side channel attacks based on power analysis. Protection from power analysis is obtained in [126] by using multi-path routing for introducing more non-determinism in the communication system.

4.2 NoC run-time monitoring

In next generation MPSoC platforms, a high number of complex concurrent applications will share the available resources providing users with new services and functionalities. At the same time, platform-based design will allow reducing the cost per single item by giving the system the possibility to easily adapt to different application re-

quirements. The complexity of these systems raises however the problem of efficiently exploiting the amount of resources available, and of understanding the system behavior once the platform has been implemented. For NoCs, new tools are therefore needed for helping designers in these tasks, exploiting information derived by measurements taken on the running system.

Modern, high performance processors deal with this problem through the use of special on-chip hardware that can monitor performances and system behavior [44]. By using a set of selected hardware units such as event detectors and counters, information about system elements is collected during program execution, analyzed, and employed to support users' applications optimization and performance improvement. In previous work, NoC monitoring was proposed for debugging and testing SoC architectures [127, 128, 129, 130, 131, 132], for detecting congestion in best-effort networks [133], for platform run-time management [134, 135], and for security purposes [17, 18]. In general, hardware monitors have been used as components of hardware and software techniques to detect, collect and interpret real-time information about system execution in order to help in testing, debugging, and validating design assumptions made on the behavior of the system and its environment [56, 57, 58].

In [129, 136], monitoring probes are inserted in the system for real-time debugging purposes. Associated programming models are discussed, as well as monitoring traffic management strategies. In [130], probes are inserted between the cores and their network interfaces. A system-level debug agent controlled by an off-chip multi-core debug controller collects information about system activities, providing in-depth analysis features such as NoC transaction analysis, multi-core cross-triggering and global synchronized time-stamping. While in [129] the system proposed requires de-packaging of the packet to retrieve the needed information, in [136] probes are located at the interface of the core, adopting therefore a solution similar to the one suggested in [130]. In [133], link utilization is monitored to implement a strategy for controlling congestion in on-chip networks.

An industrial implementation of a performance monitoring system for NoCs can be found in the Arteris NoC [137, 138]. Through *statistic collectors* and sets of *hardware probes*, embedded application software developers can analyze SoC behavior and measure performance for tuning applications and software drivers. Each statistic collector can provide up to 8 probe input channels, and monitoring results can be automatically or manually extracted for being analyzed. Type of events and monitored connections can be configured at design and run-time [138].

Compared to a similar system for high-performance processors [44], in NoCs an efficient monitoring must deal with a highly distributed environment, where event detectors and counters are distributed in different parts of the chip with often multiple clock regions. Moreover, the monitoring system should guarantee a limited intrusiveness in the detection, in the collection, and in the storage of the events [56]. In fact, in [44], the event detection is performed through a set of programmable event detectors

and counters. Data detected are collected by activating an Interrupt Service Routine which influences the program execution and interferes with the system behavior - contrary to the basic requirements for a measurement/observation system, so that the solution does not appear actually viable in a real-world environment.

Monitoring of system activities plays also an important role in the run-time management of modern adaptive and reconfigurable MPSoC platforms [134, 133, 45, 46, 47]. Having the possibility of monitoring run-time activities and reacting to unpredicted changes of the system allows dealing with the problem of designing complex system whose behavior could be only in part foreseen at design and compile time by system designers. The use of run-time adaptive systems ensures optimal, dynamic and global resource management and an enhanced predictability and use of resources [45, 134, 139].

In [53, 140], the interaction between the operating system and an NoC is studied, in particular focusing on the adaptation of system resources (i.e. injection rate of the single processing elements, routing paths used within the network) with respect to performances of applications running on the system, or for load distribution among the processing cores. In [141] run-time adaptation is employed to guarantee an adequate Quality of Service to communicating system resources while in [142, 134], the use of a resource manager is proposed to improve the utilization of resource utilization on heterogeneous MPSoCs, in particular focusing on the study of heuristics and algorithms to be used to implement the run-time management of resources and system configuration. In [133], congestion within the NoC is monitored, and injected load is adapted at run-time through a feedback control loop in order to reduce traffic within the interconnection. An overview of run-time management systems for SoCs will be presented in subsection 6.4.3.

4.3 Faults detection and fault tolerance

The range of faults that in general can affect the on-chip network infrastructure is significant and it extends from interconnects faults to logic and memory faults. For instance, examples of faults that can affect Networks-on-Chip infrastructures are crosstalk faults, memory faults in the input/output buffers of routers and network interfaces, short/open interconnect faults, or stuck-at faults affecting the logic gates of NoC routers and network interfaces [143].

The fault tolerant aspects of NoC-based systems have been recently the object of a significant amount of research effort. From the point of view of the fault model to be adopted, the NoC can be considered as a combination of several components, each one with its own fault model. It is possible to consider NoCs as composed of links, routers (or switches), and network interfaces. Every component is characterized by a different fault model which depends on the specific modules composing it, and specific fault tolerant solutions can be considered. Several fault tolerant solutions have been

proposed for NoCs, in particular addressing "hard" and "soft" faults in the links and in the router architecture.

With deep sub-micron technology and clock frequency in the GHz range, short and long distance **links** are affected by the problem of signal integrity, due to both soft errors in the data transmission, and to hard errors due to permanent faults in the device. In particular, soft errors can be caused by cross-coupling capacity and mutual inductance between wires of the links (cross-talk), degradation due to low level voltage swing in the signal transmission, alpha particles and electromagnetic effects, etc. Hard errors in interconnection can be caused by electromigration, in which metal ions migrate over time leading to voids and deposits in the wires, causing faults due to the creation of open and short circuits [144]. Depending on the specific fault, different models have been proposed for links. The Maximal Aggressor Fault (MAF) model [145] targets crosstalk effects, while a generic model based on the fault's probability of occurrence, its characteristics, and a distance matrix for the wires can be used in general for the links [146]. Short faults are moreover discussed in [147]. In the case of permanent faults, in general a faulty wire can be described as stuck-at-0 or stuck-at-1 [146]. Error control techniques [146] have been proposed as a solution to recover from faulty links, evaluating overhead in terms of area and power due to several schemes [148, 149]. In [149, 69, 150], fault tolerant solutions are proposed to mitigate transmission problems due to soft errors caused for instance by cross-talk, electromagnetic radiations, or alpha particles. Solutions proposed and evaluated are mainly based on the use of detection and correction codes [69, 150, 151], or/and retransmission [149, 152].

Routers are in general composed of first-in/first-out (FIFO) communication buffers, and several combinational blocks in charge of managing the flowing of the information, the routing of the packets, the error control, and the assignment of the available resources. Fault models obviously are strongly related to the specific implementation and functionalities offered [143]. At the functional level, several fault models have been proposed for routers and switches in NoCs. In general, in NoCs data and control faults should be considered [143, 153]. Data faults deal with the data content of the packet (payload), while control flow faults deal with the faults in information routing and in resources allocation [154]. While errors deriving from the first type of faults can be considered non-critical for the communication network, the second type of errors may lead to problems such as packet drops, lost destination, misrouting, etc. [68] and [155] present a system level fault model based on the generic properties of an NoC switch functionality. According to the model, five types of fault can occur: *Dropped Data Fault*, *Corrupt Data Fault*, *Direction Fault*, *Multiple Copies in Space Fault*, *Multiple Copies in Time Fault*. Fault tolerant design of NoC routers is the goal of [154, 156, 157]. Splitting the router operations into smaller and simpler distinct and independent modules allows the use of components with reduced logic depth. In the case of permanent failures in one of the blocks, the others will keep their function-

alities, allowing a graceful degradation of the router performance. A comprehensive router fault model is proposed, as well as safeguards to protect against various types of intra-router faults. Flick et al. [158] propose several techniques to improve fault tolerance characteristics of an NoC, including the swapping of input ports, as well as the use of crossbar bypassing and error correcting codes (ECCs) in the router data-path. In [159], default backup paths are proposed between certain router ports which serve as alternative data-paths to circumvent failed components (i.e., input buffers, crossbar switch, etc.) within a faulty NoC router.

At **network level**, proposed solutions exploit the reconfiguration capabilities of the NoC and the intrinsic redundancy of the available paths [158, 154, 160], focusing on the analysis and implementation of specific communication algorithms.In [59], a probabilistic flooding scheme based on gossip techniques is proposed. Flooding is an effective fault tolerant technique because it is highly fault tolerant. In [161], the implementation of different algorithms is compared from the point of view of the overhead in terms of area and energy. In [162], the use of multiple paths is suggested together with a methodology to guarantee in-order delivery of the packets as a solution to overcome problems in the communication due to faulty links. Related work can be also found in similar studies performed in the area of interconnection networks for distributed computing systems. Adaptive routing algorithms are suggested for avoiding faulty links or components. However, allowing the possibility to modify at run-time the path followed by packets in the network could cause deadlock situations. Algorithms based for instance on the *turn model* [163] prevents network deadlock by disallowing various network turns [164]. The turn model can be extended for *n* dimensional mesh networks to tolerate (n-1) router failures (1 router failure for 2D-mesh) [165]. An adaptive fault tolerant routing algorithm based on an odd-even turn model that addresses convex and disjoint fault regions that do not lie on the mesh boundary is discussed in [166]. Other works analyze solutions for adaptively routing around fault regions while imposing different types of restrictions on the paths [167, 168, 169, 170, 171, 172, 173, 174]. These works are mainly based on the idea of transmitting packets along the edges of the fault region encountered, and often involve disabling working IPs for meeting the shape requirements of the adaptive routing algorithm [175]. Most of the solutions proposed require the use of a large set of virtual channels for avoiding deadlock situations and for extending the use of adaptive routing to failures in fault regions of arbitrary shape.

The majority of the proposed solutions makes the assumption that the information inserted in the packets' header is correct. However, without a careful protection of the information stored into the NI and of its operations, this assumption cannot be guaranteed, thus making ineffective all the previous solutions, being in fact the information corrupted before entering the NoC. The **network interface** (NI) represents a critical element within the NoC, being in fact the separation between the cores and the rest of the system, and the ideal place for stopping propagation of cores' error. Moreover,

faults in the NI can cause the creation of corrupted messages that can disrupt the behavior of the communication network, as well as the one of the global system. Similar to routers, network interfaces are composed of several elements: input and output FIFOs, a lookup table for translating the transaction destination address to the packets' paths, several combinatorial blocks for the packet based protocol implementation, protocol adaptation, error control, and several services provided by the NI. Previous work on fault tolerance in NIs mainly focused on the definition of a functional fault model notation [153], or on providing support for errors detection in links [149, 69]. NI's faults are represented with the 2-tuple $NI(c_1, c_2)$, where c_1 is the identifier of the NI and c_2, indicates whether the NI is used as a source (S) or destination (D) of traffic [153]. In [176, 177], the use of multiple NIs for each core is proposed. Cores can be connected to more than one router, improving the fault tolerance in the case of faulty links between NIs and routers. The possibility to connect the NI to more than one router can increase the resistance to faults in the link between the NI and the router. However, the NoC will still suffer from errors in the communication due to faulty behaviors of the NI's components. Summing up, it is our opinion that the problem of faults in NIs, while very critical, has not received up to now due attention.

Chapter 5

Security in Networks-on-Chip

A security-aware design of communication architectures is becoming a necessity in the context of the overall embedded device. While the advantages brought by the use of a communication centric approach appear clear, an exhaustive evaluation of the possible weaknesses that in particular may affect an NoC-based system is still an ongoing topic. The increased complexity of this type of system can provide attackers with new means of inducing security pitfalls, by exploiting the specific implementation and characteristics of the communication subsystem.

This chapter deals with the aspects of security in NoC platforms. A first level of protection, as countermeasure against *bandwidth consumption* DoS attacks, is represented by the implementation of Quality-of-Service mechanisms in the NoC. As an example, the Aethereal NoC [94] provides a guaranteed throughput service based on time division multiplexing connections that are implemented by reserving for specific connection time slots in each router. The MANGO asynchronous NoC [178] implements instead prioritized channels over circuit switching techniques. This type of architectures allows a guaranteed bandwidth in the inter-core communication and may help avoiding malicious code running in one of the processors to flood the NoC with useless packets and requests. Other NoC architectures implementing priority-based Quality-of-Service may however still subject to the DoS attack described in subsection 3.1, in particular when the priority is determined by traffic classes [76, 96]. As pointed it out in subsection 3.1, a guaranteed-service NoC must be supported by the implementation of memory controllers providing some sort of Quality-of-Service [179], in order to avoid the memory to be the target of the DoS attack. However, the implementation of QoS techniques in the NoC does not preserve the system from DoS attacks aiming at draining the battery life.

At higher levels, security in NoC architectures and embedded systems in general can be enforced by implementing isolation techniques such as domain co-hosting or *virtualization* [119, 180]. These approaches, still not well supported and implemented in the embedded domain, present however a significant overhead, and are efficiently

applicable in the case of symmetric multiprocessor architectures, that can be controlled by a trusted entity such as the hypervisor, and that relies on the use of a memory management unit (MMU) for controlling memory accesses [119]. In the case of shared-memory heterogeneous system composed of general purpose and dedicated programmable processors, this approach is often not applicable for the non availability of MMU support for all the processing elements, as well as the lack of possibility to define privilege levels [119].

While far from proposing a comprehensive approach for handling all the possible attacks - a goal infeasible, in general, when dealing with security in computing systems - in this chapter we will focus on two specific security-related topics: the protection of data in shared memory MPSoCs, and the monitoring of NoC activities for detecting illegal behaviors.

The remainder of this chapter is organized as follows. Section 5.1 discusses contributions of this work with respect to the state of the art. Section 5.2 presents and discusses the proposed solution for securing memory accesses in NoC-based MPSoC, in particular focusing on the key element of the proposed architecture, i.e., the *Data Protection Unit*. Section 5.3 discusses a system for the run-time configuration of the DPUs, while section 5.4 presents implementation and experiment results. In section 5.5, we extend the work performed in the first sections of the chapter by describing the basic elements of a security monitoring system for NoCs. Section 5.6 focuses on the implementation details of the components of the monitoring system, while section 5.7 presents implementation details.

5.1 Contributions with respect to the state of the art

Aspects of security related to NoC-based systems have been taken into account by the research community only recently. As discussed in chapter 4, previous work focused on the exchange of sensitive information on-chip through encrypted communication, on securing NoC based reconfigurable systems, and on supporting the creation of secure and non-secure areas in bus based SoCs. With respect to the related work referenced in section 4, the solutions discussed in this chapter can be considered orthogonal and complementary.

On the one hand, we propose a solution that represents a step further with respect to previous implementations of data protection techniques, since, for the first time, it faces the problem of data protection on an NoC-based Multiprocessor System on-Chip. With respect to previous work on data protection, our technique has finer granularity (especially with respect to the ARM TrustZone), which will be more useful for the next generation of security-enhanced multiprocessor systems. Moreover, the binary co-hosting implemented by the TrustZone architecture is no more sufficient to satisfy security requirements of new complex architectures. With respect to a bus-based solution, our work takes into account the characteristic of NoC-based systems of

representing a distributed environment. The memory protection solution proposed can be used to support co-hosting and virtualization, by providing a programmable physical separation logic effective in protecting important resources of the base domain (or hypervisor) from, for instance, downloaded applications on other virtualized domains [180].

On the other hand, our work on secure monitoring represents a first attempt to discuss in detail characteristics and implementation costs as well as design trade-offs of including such a type of system in NoCs, thus taking a step forward towards the realization of a security solution at system level that could adopt approaches similar to the Intrusion Detection Systems (IDSs) employed in data network security [181].

To summarize, the main contributions described in this chapter are:

- The proposal of a data protection hardware module (*called Data Protection Unit (DPU)*) that enforces access control on protected memory blocks, and that guarantees secure accesses to memories and/or memory-mapped peripherals. Access to a given memory space is granted only if the initiator of the request is authorized to perform the operation requested. Access filtering is performed by considering memory address, operation requested, and status of the initiator.

- The proposal of a system for managing the run-time configuration of the several DPUs distributed in the NoC.

- The proposal and evaluation of a monitoring system for helping detect attacks aimed at retrieving sensitive information from the system or at causing Denial-of-Service (DoS) by exploiting implementation characteristics of the NoC.

5.2 Secure memory accesses in NoCs

This section describes an original data protection infrastructure for architectures adopting the Network-on-Chip communication paradigm. The proposed infrastructure provides support for secure accesses to memory locations in shared memory multiprocessor NoC architectures. In particular, as reference target system, we refer to the NoC and NI introduced in chapter 3, i.e., a communication infrastructure implementing a transaction-based protocol within memory-mapped components, and an OCP compliant NI.

In this section, we present a solution to contrast attacks that aim at obtaining access to restricted blocks of memory, and that exploit techniques (such as the buffer overflow) that have been the basis for a relevant number of attacks and the most common form of security vulnerability for the last ten years, as discussed in chapter 3. The secure network architecture proposed in this section is based on the use of Data Protection Units (DPUs) integrated into the network interfaces of the NoC, which

guarantee secure accesses to memories and/or memory-mapped peripherals, by enforcing access control rules specifying the way in which an IP initiating a transaction to a shared memory in the NoC can access a memory block. The partitioning of the memory into blocks allows separating sensitive and non-sensitive data for the different processors connected to the NoC. The proposed DPU represents a hardware solution enabling access to a given memory space only if the initiator of the request is authorized. Access filtering is performed by considering not only the memory address but also the type of the requested operation (data load/store and instruction execute), and the status of the initiator (user or supervisor mode, secure or unsecure mode[8]). Use of the Data Protection Unit offers the possibility to easily load/store critical data and instructions while protecting them from illegal accesses by malicious code running on compromised cores, without requiring time-consuming encryption/decryption and thus guaranteeing a fast memory access. The run-time configuration of the programmable part of the DPUs is managed by a central unit, the Network Security Manager (NSM), which guarantees the run-time flexibility of the proposed approach. Two different basic DPU architectures will be evaluated, which provide different trade-offs in terms of implementation costs: the case of the DPU implemented at the target, and its implementation at the initiator.

In the first proposed architecture (see figure 5.1(a)), the DPU is a module embedded into the NI of the target memory (or the memory-mapped peripheral) to be protected, supplying services similar to those offered by a traditional "firewall" in data networks. In briefly, the DPU *filters* the requested accesses to the memory blocks through a lookup of the access rights, done in parallel with the protocol translation within the network interface. In the second proposed solution, the filtering of memory accesses is done at the network interface of each initiator (figure 5.1(b)). A detailed description of the behavior of the protection system is given in the following subsections, where we describe the two alternative implementations of the proposed solution for data protection: the DPU architecture implemented at the target NI, called *DPU@TNI* (section 5.2.1), and the DPU architecture implemented at the initiator NI, called *DPU@INI* (section 5.2.2).

5.2.1 DPU at the target NI (*DPU@TNI*)

Figure 5.2 and figure 5.3 respectively show the architecture details and the whole system overview when the DPU is embedded in the target NI (*DPU@TNI*). For this architecture, the DPU checks the header of the incoming packet to verify if the requested operation is allowed to access the target. In order to implement this type of architecture, we need to refer to an NoC packet such as the one shown in figure 5.4. In fact, the DPU exploits the information forwarded within the packet to perform access control on the requests arriving at the target. However, the proposed DPU architecture can be easily adapted to different packet formats. The proposed format of the packet header (see figure 5.4) is compliant with the OCP interface and it is composed of the

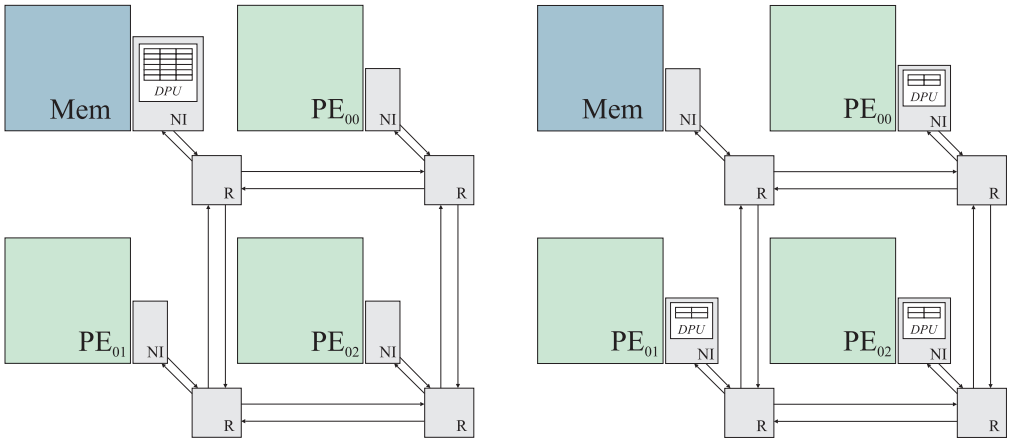


Figure 5.1. Simple system with three initiators (PE_i) and one target (*Mem*) showing the two different network architectures using the DPU (a) at the target NI and (b) at the initiators NIs.

following fields:

- *DestID* is used to identify the target of the request (we assume a table-based routing depending on source and destination addresses). The OCP *MAddr* field is converted by the NI, following the shared-memory abstraction, to obtain the *DestID* identifier;
- *SourceID* identifies the initiator of the transaction and, depending on the granularity of the system, could refer to the identification number of the node in the NoC, to a single IP in a cluster (assuming more IPs connected to the NoC through the same NI) or to a thread running on a specific IP core. This field comes from information provided by the OCP interface (*MConnID* and *MthreadID*) representing respectively the processor identifier and the thread identifier, and a value stored into the NI representing the network node identifier (*NodeID*);
- *MemAddr* is the memory address of the initiator requesting access and it is the direct translation of *MAddr* signals of OCP interface;
- *Length* field represents the length of the data to be sent/retrieved. The burst information derived by the OCP/IP interface (*MBurstLength*, *MBurstPrecise* and *MBurstSeq*) are used to define the length of the information to be sent/received (expressed as number of words);
- *L/S* encodes which kind of operation ($\overline{load}/store$) the initiator requests at the target memory address. It corresponds to the *MCmd* signal of the OCP interface.

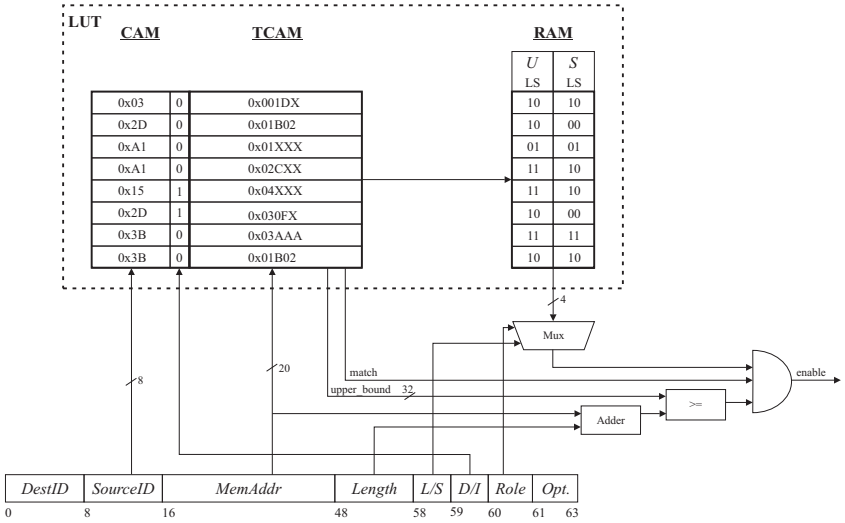


Figure 5.2. DPU architecture at the target network interface (*DPU@TNI*).

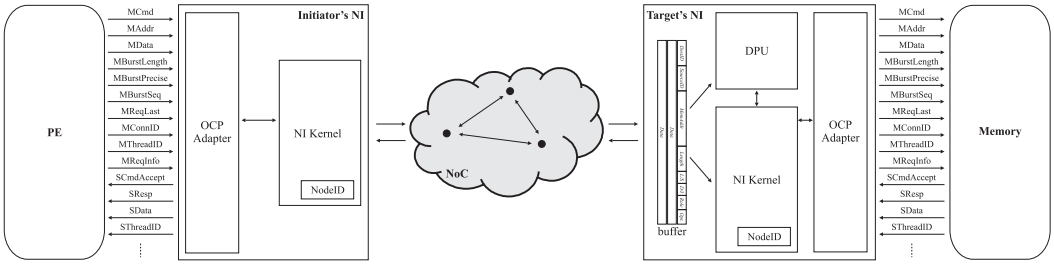


Figure 5.3. Overview of the whole NoC-based architecture including the *DPU@TNI*.

- *D/I* and *Role* respectively identify $\overline{data}/instruction$ and initiator role ($\overline{user}/supervisor$) associated with the request. These bits correspond to *MReqInfo[2:3]* signals that, following OCP recommendations for the security profile [8], are used to forward information about type of data and initiator role;
- *Opt* represents an optional field that can be used to add further network services.

The size of *DestID*, *SourceID* and *Length* fields depends on system parameters (number of targets for *DestID*, number of initiators and threads for initiator for *SourceID*, and maximum burst size for *Length*). As an example, the size of the *SourceID* fields has been designed in figure 5.4 for a system with a number of connections up to 255, where the connection number is a combination of the number of network nodes, number of IPs on the same network node and maximum number of threads active in the same IP.

This access control in the DPU@TNI is done mainly at the time of the reception of a new packet by using a lookup table (LUT), where entries are indexed by the

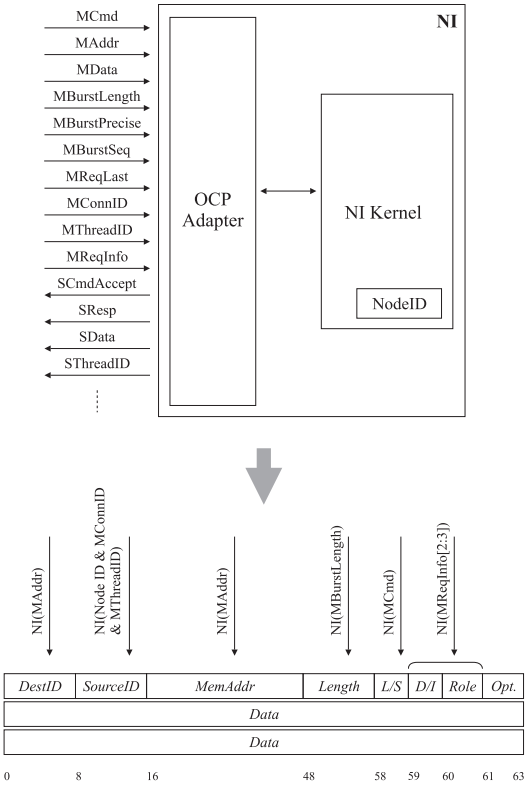


Figure 5.4. Interface between the OCP signals and the packet format used within the NoC.

concatenation of the *SourceID*, the type of information ($\overline{D}/I$), and the starting address of the requested memory operation *MemAddr*. The number of entries in the table depends on the number of memory blocks to be protected in the system, as well as on the number of initiators. In the implementation shown in figure 5.2, we assume 4KB as the size of the smallest memory block to be managed for the access rights. This means that all data within the same block of 4KB have the same rights (corresponding to the 12 LSB in the memory address) and that we use only the 20 most significant bits of the *MemAddr* field for the lookup.

The lookup table of the DPU is the most relevant part of the architecture and it is composed of three parts:

- A Content Addressable Memory (CAM) [182] used for the lookup of the *SourceID* and type of data ($\overline{D}/I$);
- A Ternary Content Addressable Memory (TCAM) [182] used for the lookup of the *MemAddr*. With respect to the binary CAM, the TCAM is useful for grouping ranges of keys in one entry since it allows a third matching state of 'X' (*Don't Care*) for one or more bits in the stored datawords, thus adding more flexibility to

the search. In our context, the TCAM structure has been introduced to associate to one LUT entry memory blocks larger than 4KB.

- A simple RAM structure used to store the access rights values.

Each entry in the CAM/TCAM structure indexes a RAM line containing the access rights ($allowed/\overline{not\ allowed}$) for user load/store and supervisor load/store. The type of operation ($\overline{L}/S$) and its role ($\overline{U}/S$) taken from the incoming packets are the selection lines in the 4:1 multiplexer placed at the output of the RAM. Moreover, a parallel check is done to verify that the addresses involved in the data transfer are within the memory boundary of the selected entry.

If the packet header does not match any entry in the DPU, there are two possible solutions, depending on the security requirements. The first one is more conservative (shown in figure 5.2), avoiding access to a memory block not matching any entry in the DPU lookup table by using a *match* line. The second one, less conservative, enables the access also in the case when there is no match in the DPU lookup table. This corresponds to the case when a set of memory blocks does not require any access verification.

The output enable line of the DPU is generated by a logic AND operation between the access rights obtained by the lookup, the check on the block boundaries and, considering the more conservative version of the DPU, the *match* on the lookup table.

Given the complexity of the protocol conversion to be done by the NI kernel, we can assume that the DPU critical path is shorter than the critical path of the NI kernel (as confirmed by the results reported in section 5.4). Under this assumption, integrating the DPU at the target NI guarantees that no additional latency is associated with the access rights check since, as shown in figure 5.3, the protocol conversion and the DPU access are done in parallel.

5.2.2 DPU at the initiator NI (*DPU@INI*)

Figure 5.5 and figure 5.6 show respectively the architecture details and the whole system overview when the DPU is embedded at the initiator NI (*DPU@INI*). For this architecture, the DPU directly uses the signals coming from the OCP slave interface of the NI for the access rights evaluation, and no special packet format is needed for this type of implementation of the DPU. As shown in figure 5.4, when the PE initiates a transaction driving the OCP Master interface signals, the NI looks up the *MAddr* signals in order to obtain the routing information to be inserted in the *DestID* field in the header of the packet. At the same time, the DPU looks up the information coming from the OCP interface to check if the request has the right to access the addressed memory block. As shown in figure 5.5, in the *DPU@INI* architecture each entry of the LUT is indexed by the concatenation of *MConnID*, *MThreadID* and part of *MAddr*. In this case, the number of entries in the LUT depends only on the number of blocks to be protected

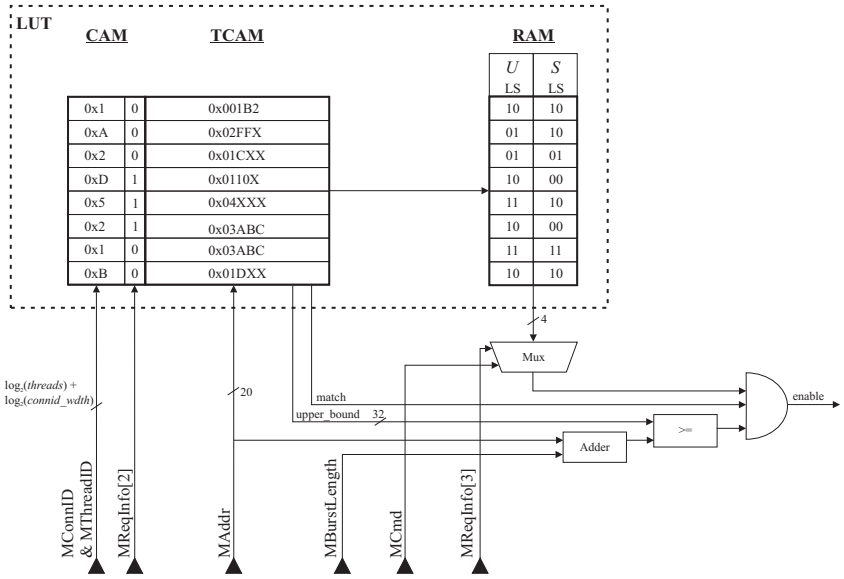


Figure 5.5. DPU architecture at the initiator network interface (*DPU@INI*).

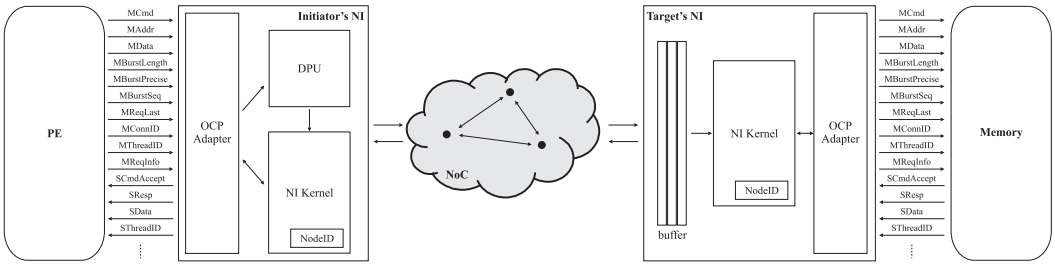


Figure 5.6. Overview of the whole NoC-based architecture including the *DPU@INI*.

in the target memories since the initiator NI is related to only one processing element. As well as in *DPU@TNI*, when the information present at the OCP interface matches one entry line in the LUT, the LUT returns the access rights for the roles of the initiator (user load/store and supervisor load/store). The *MCmd* and *MReqInfo* signals coming from the OCP interface represent the selection lines in the 4:1 multiplexer placed at the output of the RAM. A parallel check on the *MBurstLength* is done to verify if the block boundaries are respected.

As mentioned for the *DPU@TNI*, given the complexity of the protocol conversion to be done by the NI kernel, we can safely assume that the DPU critical path is shorter than the critical path of the NI kernel (as confirmed by the results reported in section 5.4). Under this assumption, integrating the DPU at the initiator NI guarantees that no additional latency is associated with the access right check since, as shown in figure 5.6, the protocol conversion and the DPU access are done in parallel.

With respect to *DPU@TNI*, the overhead associated with the dimension of the CAM is smaller because it is not necessary to include in the LUT the identifier of the initiator node (*NodeID*).

Checking the access rights at the initiator NI avoids initiating the network transaction whenever the request would be rejected. As a matter of fact, with respect to *DPU@TNI*, the *DPU@INI* blocks bad memory requests before they enter the network; this results in no NoC traffic and less energy consumption. As drawback of *DPU@INI* solution, when a target would not require data protection, we pay for DPU accesses in all instances they would not be necessary. In fact, since the protocol translation (thus the target identification) and the DPU access are done in parallel to hide the latency overhead, it is not possible to avoid DPU accesses such targets that not require data protection. As it will be shown in section 5.4, the choice of the best DPU architecture to implement depends on the target system.

5.2.3 Example of data transfer in systems adopting the DPU

This subsection outlines through an example the steps performed during a memory transfer taking as reference the system shown in figure 5.3 (*DPU@TNI*). In the example, we consider a *store* operation of 512 Bytes of data, starting from the memory address equal to *0x01B02CF0*. The PE is identified by a *SourceID* equal to *0x2D*, its operating role is *user* (*Role*=0), and the target memory has a node identifier (*DestID*) equal to *0x0A*.

In the case of a *store* operation, the PE drives the OCP signals to begin the data transfer in bursty way [8] and the NI behaves as slave for the OCP protocol. As described in section 5.2.1, the front-end of the NI looks up the information on the first address of the burst on *MAddr* to obtain the routing information to be inserted in the field *DestID* in the header of the packet. The control signals of the OCP transaction are extracted from the OCP adapter and coded into the NI kernel to build the packet header (see figure 5.4). Then, the NI samples data from OCP *MData* and create the payload of the packet.

Considering the *DPU@TNI* architecture, the header of the packet, followed by the payload is therefore sent through the network to the target NI. While the NI Kernel is processing the header of the packet, and waiting for the payload, the DPU checks in parallel the header information to verify the access rights.

As shown in figure 5.7, since one of the LUT entries matches the request, the corresponding 4 bits stored in the RAM are forwarded to the multiplexer. In parallel, the DPU also verifies that the addresses involved in the data transfer (from *MemAddr* to *MemAddr*+512) are within the memory bounds of the selected entry (*0x01B02000-0x01B02FFF*). Because the header information matches one entry of the LUT, the related right value is equal to '1', and the memory addresses are within the boundaries, the DPU enables the requested *store* operation, allowing also the NI to initiate the OCP

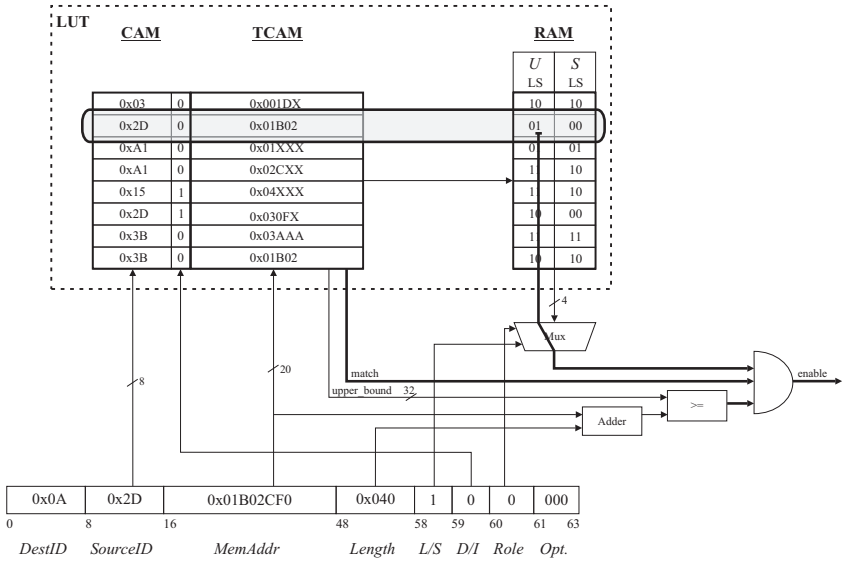


Figure 5.7. Example of usage of DPU@TNI for a store memory access.

transaction for a *write* in the target memory. When the transaction completes, a signal of acknowledgement is sent back to the initiator NI to notify the end of the data transfer. In the case of not allowed memory request, the DPUs would stop the protocol translation avoiding the OCP transaction, notifying a security warning and sending back to the initiator NI a not acknowledgement message.

The same example can be easily replicated by considering the system outlined in figure 5.6 (*DPU@INI*). The only significant difference is that being the lookup at the initiator NI, while the NI kernel is processing the OCP control signals to create the packet header, the DPU checks in parallel these signals to verify the access rights, blocking (or not) the packet generation.

5.3 Run-time Configuration of the DPUs

In this section, we extend the solution presented in the previous section to program DPUs at run-time. This feature allows the system to optimize the use of the protection units and to modify them in the case of changing application-scenarios. To configure the DPUs distributed in the system, we adopt a centralized approach. An overview of the overall system proposed to program the DPUs at run-time is shown in figure 5.8, where the main elements are the Network Security Manager (NSM) and the DPU@TNI. The proposed run-time configuration can be applied also to DPU@INI. The NSM is in charge of accepting or refusing new restriction rules on the memory blocks and configuring appropriately the DPUs.

In this section, first we discuss the configuration of the system, focusing in particu-

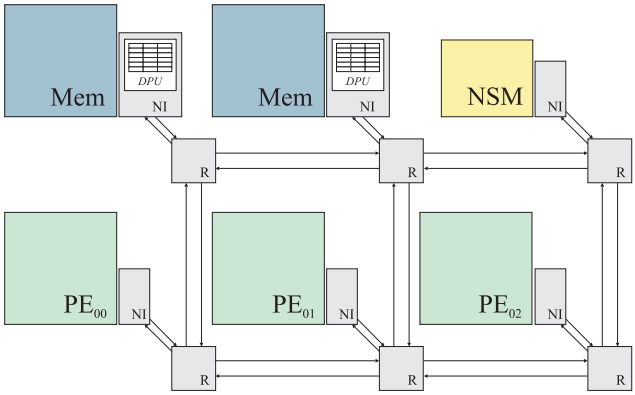


Figure 5.8. System architecture including the Network Security Manager (NSM) to program the DPUs at run-time.

lar on the activities involving the initiators of the transactions and the NSM. Secondly, we focus on the DPUs, in particular analyzing the architectural changes necessary to support run-time configuration and the interaction with the NSM. Thirdly, some possible security faults of the system and countermeasures are presented, especially to counteract *spoofing attacks*.

5.3.1 Network Security Manager

The NSM is a dedicated IP block that, together with the DPUs, dynamically associates access rights with selected memory spaces. The possibility to dynamically manage the memory protection is useful in architectures where the number, position and/or size of the memory regions to be protected cannot be resolved at design time.

Since we possibly do not want every processing element to be able to request the assignment of access rights to the NSM, these service request messages are filtered by the NSM considering only those arriving from particular processing elements in a particular *state* and performing specific *roles*. In detail, only initiators in *secure* status and acting as *supervisor* are enabled to communicate with the NSM. According to OCP specifications, this is translated into the possibility for the initiator to set to '1' the values of specific bits in *MReqInfo*. Obviously, the access to these bits must be restricted to initiators acting in the previously specified operating mode.

When an application requires a memory space to be protected, the right assignment operation is issued by the initiator to the NSM. If that space is not yet under rights management, the request is accepted and a message is sent to update the DPUs involved in the process. The NSM keeps track of all successful protection requests in a record for managing successive requests. Each entry of the record contains the information regarding the access rights and the DPUs involved in the request. If the memory space requested to be protected is already under rights management, and the

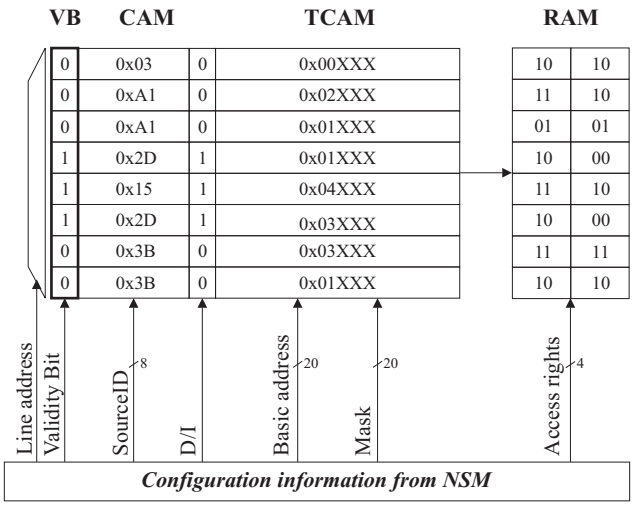


Figure 5.9. LUT modified to support run-time configuration.

requesting initiator is the owner of the block, the request is accepted and the corresponding record is updated. If the requesting initiator is not the memory block owner, the request is rejected by the NSM. The transaction related to the rights assignment can be considered completed when all the DPUs involved in the update acknowledge to the NSM.

When a protected region needs to be released from the access restriction, the request is sent to the NSM. If the source of the request is the owner, an update message is sent to the DPUs and only when all responses are received by the NSM, the protected region can be considered released. If the requesting initiator is not the memory block owner, the request is rejected by the NSM.

Beside the dynamic configuration of the DPUs, the Network Security Manager could be used more in general to reprogram all the programmable parts of the NoC, such as the network interface registers.

5.3.2 DPU to support run-time configuration

The modified DPU architecture supporting dynamic reconfiguration is shown in figure 5.9. We consider, without loss of generality, only the architecture previously named as *DPU@TNI*. However, similar considerations apply to *DPU@INI*.

As shown in figure 5.9, a *Validity Bit* (*VB*) must be added to each line of the DPU. Only the entries of the lookup table with a *Validity Bit* equal to '1' are taken into consideration when checking the access rights, while the others are ignored. To configure at run-time the protection module by writing the necessary data into the lookup table, a port has been added to the DPU. The NI hosting the DPU manages the updates for the DPU, processing the requests coming from the NSM.

When the NSM requests the storage of a new access restriction to the DPU, it communicates to the hosting NI the address of the lookup and the information to be inserted in the DPU lookup table. To avoid conflicts between the requests of the NSM and the packets being processed by the DPU, priority is given to the former (configuration requests). In the case of access requests arriving during the configuration of the DPU, they will wait in the NI's input buffers until the lookup table has been updated. After successful completion of the update of the DPU, the NI enables the processing of the requests coming from the initiators and sends back to the NSM a signal of acknowledgement.

In the case of deletion of one of the access restrictions, the NSM sends a request to release the corresponding line of the lookup table. The NI processes the request and notifies to the NSM the completion of the transaction.

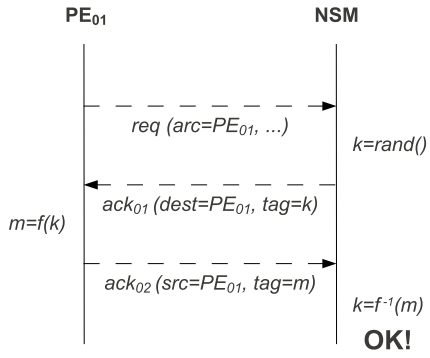
The overhead required by the dynamic reconfiguration of the DPU occurs only when the memory protection must be updated. However, this situation usually happens only once per application run. The possibility to dynamic reconfigure the DPUs increases the flexibility of the system, but its variable overhead, involving several network messages, affects the time determinism of an application/context switching. For hard real-time systems, the use of this feature should be avoided.

5.3.3 Possible security faults and countermeasures

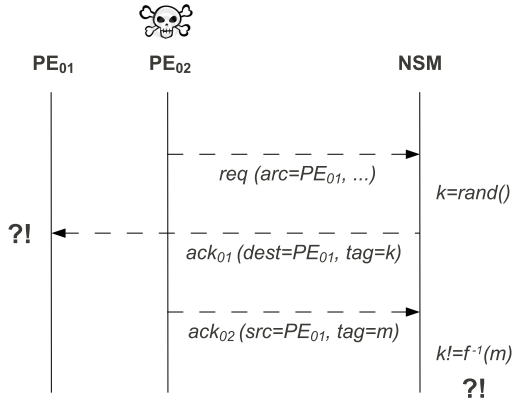
In this section, we consider a possible security fault that may affect the type of dynamic data protection mechanism we described, and we propose a countermeasure. In particular, we focus our attention on those parts of the system - such as registers - that can be modified by the applications, and whose unauthorized or careless accesses could compromise the effectiveness of the NSM and the DPUs.

As described in section 5.2, the identifier of a request (*SourceID*) is mainly based on the identifier of the NI (*NodeID*) and the identifier of the connection (*MConnID*). The possibility to maliciously modify the information stored in those registers could imply the hiding of the identifier of the processor requesting the access to the NSM or to the protected memory blocks, by substituting it with that of an authorized processor with higher privileges. This fact could cause a serious security problem since, as an example, it opens the possibility to access the NSM and to reprogram all the DPUs, enabling the access to protected locations. In the context of network security, these attacks are called *spoofing attacks* [183]. To carry out this kind of attack, an attacker successfully masquerades as a legitimate party by falsifying data and thereby gaining an illegitimate access.

In such NoC implementations where the mentioned registers are not hardwired or efficiently secured, in order to avoid unauthorized accesses to the NSM, we propose a possible countermeasure to *spoofing attacks*. In figure 5.10(a) we show a simple double acknowledgement authentication protocol to overcome spoofing attacks. The protocol attempts to identify a malicious substitution of the processor identifier by sending an



(a) Authentication protocol



(b) Attack

Figure 5.10. Overview of the authentication protocol between an initiator and the NSM (a) and the identification of a possible attack (b).

acknowledgement message to the source of the request and requiring a correct answer, as shown in figure 5.10(a). In this way, if the real source of the request is not the one written in the original request, the malicious processor cannot end the authentication protocol, since it does not receive the acknowledgement message ACK_01. To avoid also the possibility that the malicious processor could answer the NSM without having received the ACK_01 (as in figure 5.10(b)), the ACK_01 message contains also a *TAG* field that should be elaborated by the source (by using a simple translation function f) and sent back to the NSM that will check the correctness. In fact, the attacker, represented in figure 5.10(b) by PE_02, attempts a spoofing attack by masquerading its real identity with the one of PE_01 in the request ($SRC = PE_01$). The NSM sends the ACK_01 response to PE_01 including the *TAG* value k (since it is the requester in the message) but PE_02 cannot know what is the *TAG* value of the response, k. Even though PE_02 does not know the real *TAG* value (as shown in figure 5.10(b)), it attempts to end

the authentication process by sending a random tag m. This tag value has a probability equal to $1/2^n$ (where n is the number of bit of the *TAG* field) to be equal to the correct value. During this failed authentication, two security warnings are reported: the first one given by $PE_0 1$ that receives the $ACK_0 1$ message without any previous request and the second one given by the NSM due to the *TAG* value mismatch.

The same problem as that just shown for access to the NSM could arise for access to memory locations protected by using DPUs, or for reconfiguration of DPUs, or more in general for the reprogramming of all the programmable parts of the network. We suggest two proposals to avoid *spoofing attacks* also in these cases by using an authentication mechanism:

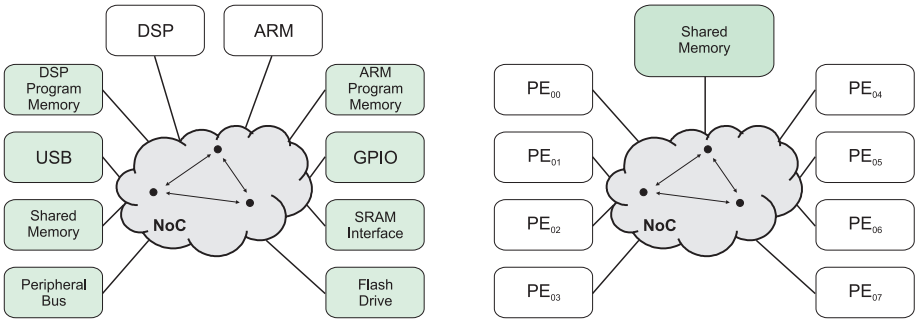
- The first solution takes into account the same protocol, based on a double acknowledgement as for the NSM, for all restricted accesses. In this case, the restricted request is issued to the target passing through the DPU that will perform the authentication protocol; the target will send the $ACK_0 1$ message together with the *TAG* information, and will wait for the $ACK_0 2$ message from the initiator to check its identity.

- The second solution uses the double acknowledgement protocol only to exchange a key, while the other accesses will be authenticated by appending the key to the request. To increase the level of security, we can restrict the use of a key by using the concept of *session*. A session represents a time-slice, defined in terms of fixed number of operations or fixed time duration, during which we can use the same key. When the session ends, a new key needs to be exchanged for the management of the accesses in the next session.

Selecting the solution to be adopted strongly depends on several factors, such as the level of security that we want to enable, the frequency of the accesses to protected location, the position of the DPUs into the network and the available overhead to meet the performance constraints.

5.4 Experimental Results

In this section, we discuss the experimental results related to the proposed data protection mechanism for NoC-based architectures, evaluating the overhead introduced by DPU modules into the communication subsystem. In particular, section 5.4.1 presents the experimental setup, while section 5.4.2 and 5.4.3 respectively show the synthesis results for the two different implementations of the DPU and the overhead introduced by the data protection mechanism for two case studies.



(a) Arch1 (b) Arch2

Figure 5.11. Architecture overview of the two case studies. Targets and initiators are in gray and white boxes respectively.

5.4.1 Experimental Setup

To evaluate the impact introduced by the proposed data protection mechanism into a System-on-Chip architecture based on NoC, we considered two systems representing a typical embedded architecture and a shared-memory multiprocessor. In particular, the DPU overhead has been evaluated in the context of the NoC subsystem. The two case studies are shown in figure 5.11:

- *Arch1* represents a typical embedded architecture with 2 initiators and 8 target-memories/peripherals. For our purposes, we consider each target of the memory space partitioned into 4 blocks.

- *Arch2* represents a shared memory multiprocessor composed of 8 initiators and 1 target-memory partitioned into 16 blocks.

Within the NoC clouds represented in figure 5.11 we consider routers in a mesh topology, each connected with a network interface. The type of the NI (initiator or target) depends on the type of the connected IP module. Both case studies are used to show how, by varying system characteristics, the best DPU configuration and its energy/area overhead on the communication subsystem will vary as well. Concerning performance results for the two case studies, we will show in section 5.4.2 how no performance overhead is introduced by the DPU because the DPU critical path is less than the NI kernel critical path. Given that, the system-level performance is not impacted by the DPU insertion.

Regarding the evaluation of the a single DPU module, synthesis and energy estimation have been respectively performed by using Synopsys Design Compiler and Prime Power with 0.13μm HCMOS9GPHS STMicroelectronics technology library.

To compare area and energy overheads introduced by the proposed data protection mechanism into the NoC subsystem, table 6.3 shows area and energy values for each

Table 5.1. Area and energy dissipation due to the NoC components considering a 32-bit data-path running at 500MHz.

	NoC router			NI-initiator		NI-target	
	3p	**4p**	**5p**	**arch1**	**arch2**	**arch1**	**arch2**
Area [mm^2]	0.078	0.112	0.143	0.141	0.166	0.172	0.172
Header Energy [pJ]	75.2	80.1	88.1	101.2	92.7	107.3	107.3
Payload Energy [pJ]	63.3	65.1	67.8	44.7	44.7	52.3	52.3

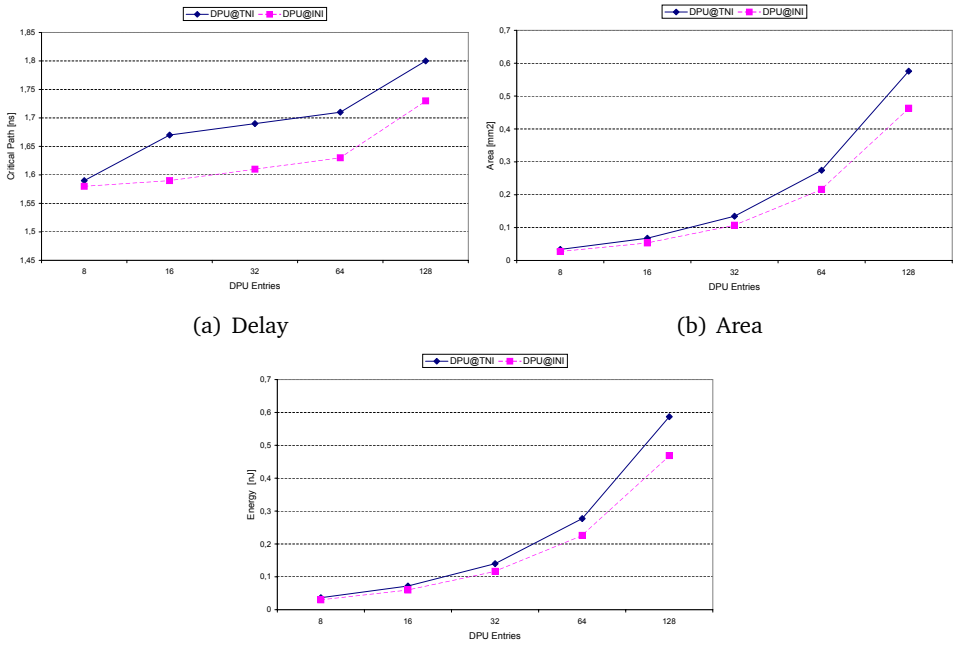
NoC components, obtained by using the PIRATE-NoC compiler [184]. All results presented in table 6.3 are obtained by considering a frequency of 500MHz imposed by the critical-path of the NI-kernel both at the initiator and target. The NIs implement the OCP interface. The router adopts a wormhole control flow strategy, it includes input and output buffers, a three-stage pipeline, and table-based routing [184]. The results on table 6.3 have been generated by considering a target network characterized by a 32-bit data-path, with the router input/output buffer depth equal to 4 and the target and initiator NIs buffer sizes equal to 16 and 8 respectively. Since we considered a mesh topology, table 6.3 shows the area and energy values considering the three possible router configurations in terms of number of ports (3p, 4p and 5p), including the NI port. Table 6.3 also shows the values for each NI (initiator and target) for the two architectures (*Arch1* and *Arch2*). Area and energy values for the NI-initiator depend on the target architecture because the logic for the generation of the *DestID* field depends linearly on the number of targets in the system. According to [184, 185, 186], the energy consumption of the different network units must consider two different costs for the header flit of the packet and for each flit of the payload. The energy associated with the header flit is higher than that of the payload, since the header flit activates all the control parts of the network (e.g. routing unit and protocol translation).

5.4.2 DPU Synthesis Results

In this section, we present the synthesis results for the two proposed implementations of the DPU architecture (*DPU@TNI* and *DPU@INI*) presented in section 5.2.

Figure 6.11 shows the synthesis results for a single DPU module in terms of delay [ns], area [mm^2] and energy [nJ] by varying the number of entries for the *DPU@TNI* and *DPU@INI*. All the results have been obtained by targeting the synthesis to a clock frequency of 500MHz imposed by the critical path of the NI-kernel. As shown in figure 5.12(a), the critical path of all the explored DPU configurations (up to 128 entries) are below 2ns, confirming that the DPU does not introduce any additional cycle to each memory request.

Figure 5.12(b) shows that for both architectures (*DPU@TNI* and *DPU@INI*) the DPU area increases almost linearly with the number of entries. This is due to the fact that the most significant area contribution is given by the CAM/TCAM included



(a) Delay

(b) Area

(c) Energy

Figure 5.12. Synthesis results in terms of delay, area and energy by varying the DPU entries for *DPU@TNI* and *DPU@INI*.

Table 5.2. DPU area comparison with respect to ARM920T processor and 16KByte SRAM memory.

ARM920T $[mm^2]$	**16KB SRAM** $[mm^2]$	**128-entry DPU** **@TNI** $[mm^2]$	**@INI** $[mm^2]$
4.70	2.31	0.57	0.46

in the DPU. Although the two versions of the DPU behave similarly, the area value of *DPU@INI* are always smaller than those of the *DPU@TNI* with the same number of entries. This gap is due to the different type of interfaces (OCP or packet-based) and to the reduced number of CAM bits.

As expected, since the main part of the DPU is composed of a CAM/TCAM, the energy trends shown in figure 5.12(c) by scaling the number of DPU entries are similar to those already described for the area values.

To evaluate the complexity of DPU architectures in a System-on-Chip context, in table 5.2 we compare the area values with two widely used IPs for SoC architectures: an ARM920T processor with 16KByte of Instruction and Data Cache running at 250MHz and a 16KByte SRAM memory with 16 Byte for each entry and 1 read and 1 write port. The comparison has been done by considering the same technology (0.13μm) for all

Table 5.3. Area and energy overhead due to Data Protection Units for the two case studies.

	Number of DPUs	**DPU Area** $[mm^2]$	**Total DPUs Area** $[mm^2]$	**Energy for DPU Access** $[pJ]$
Arch1				
DPU@TNI	8	0.034	0.268	36.5
DPU@INI	2	0.111	0.221	117.4
Arch2				
DPU@TNI	1	0.581	0.581	508.6
DPU@INI	8	0.055	0.443	59.9

IPs. For both DPUs we selected the LUT with 128 entries. With respect to the area of ARM920T processor [187] and to the area of 16K SRAM memory [188], the area of *DPU@TNI*/*DPU@INI* is respectively the 12%/10% and 25%/20%. A further comparison with the components of the NoC subsystem will be shown in the next subsection for the two selected cases studies.

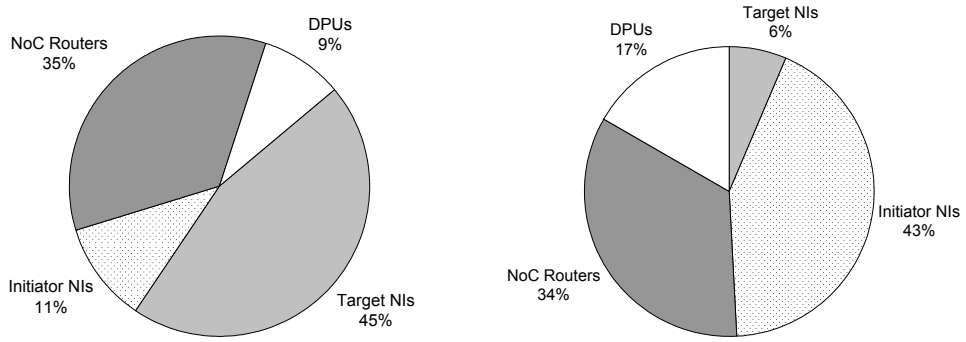
5.4.3 Case Studies

Given that no performance overhead has been introduced by the proposed DPU, in this section we evaluate the area and energy overhead introduced by the proposed data protection mechanism into the communication subsystem for the two case studies shown in figure 5.11.

Table 6.1 shows area and energy overhead due to both DPUs, applied to all the memory elements and peripherals of the system. The column *DPU Area* reports the area overhead of each DPU, while the column *Total DPUs Area* reports the sum of the area of all the DPUs distributed on the different NIs. The column *Energy for DPU Access* represents the cost for access to the DPU for each memory request.

Considering *Arch1*, the area of a single *DPU@INI* is larger than *DPU@TNI*, since the number of entries is 32 (4 memory blocks × 8 targets) in the former and 8 (4 memory blocks × 2 initiators) in the latter. Similarly, the energy for DPU access is larger for *DPU@INI* than for *DPU@TNI*. Globally, for *Arch1* there are 64 DPU entries (4 memory blocks × 8 targets × 2 initiator) distributed on the system for both DPU solutions. Since the number of bits for each DPU@INI entry are smaller than those for the DPU@TNI and the DPU area is mainly due to the LUT, the *Total DPUs Area* for the DPU@INI solution is smaller than in DPU@TNI solution.

Considering *Arch2*, the area of a single *DPU@TNI* is larger than *DPU@INI* since the number of entries is 128 (16 memory blocks × 8 initiators) in the former and 16 (16 memory blocks × 1 target) in the latter. Similarly, the energy for DPU access is larger for *DPU@TNI* than for *DPU@INI*. Globally, for *Arch2* there are 128 DPU entries (16 memory blocks × 1 targets × 8 initiator) distributed on the system for both DPU



(a) Arch1 (b) Arch2

Figure 5.13. Area breakdown for the NoC subsystem including DPUs.

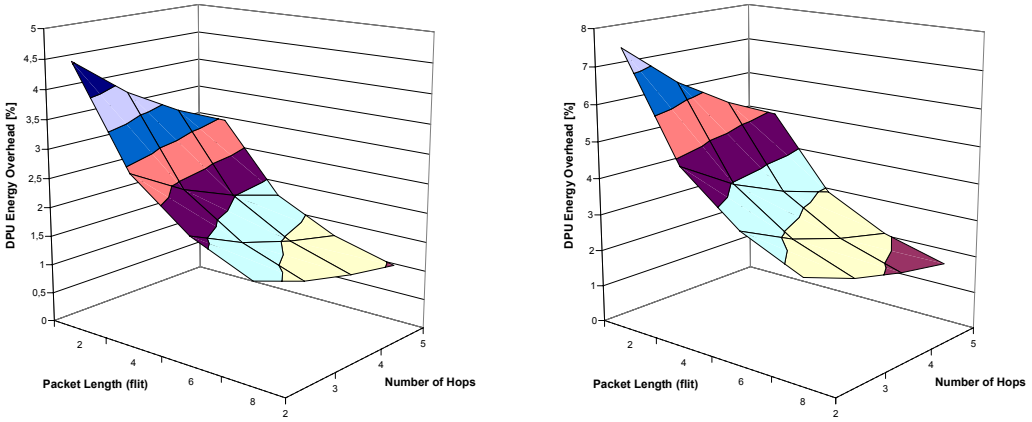
solutions. As before, since the number of bits for each DPU@INI entry are smaller than those for the DPU@TNI and the DPU area is mainly due to the LUT, the *Total DPUs Area* for the DPU@INI solution is smaller than DPU@TNI solution.

In the following analysis, we consider only one DPU solution for each case study. While for Arch2 we selected the *DPU@INI* architecture due to its better results in terms of both cost functions (*Total DPUs Area* and *Energy for Access*), for Arch1 we selected the DPU@TNI, as shows a limited area overhead but a reduced energy cost for access.

For both cases, figure 5.19 shows the area breakdown of the NoC subsystem in terms of NoC routers, target and initiator network interfaces (see table 6.3) and DPUs (see table 6.1). The area overhead introduced by the DPUs is limited to 9% and 17% with respect to the NoC subsystem for *Arch1* and *Arch2* respectively. The difference in the number of network targets and initiators in the two architectures is reflected in the area breakdown: the target/initiator NIs rate results 45%/11% with respect to 6%/43% for *Arch1* and *Arch2* respectively.

Concerning energy, figure 5.14 shows the DPU energy overhead with respect to the energy consumed by the NoC subsystem for a memory access. Because the energy consumed by the NoC for a memory access depends on the length of the packet and on the network distance (number of hops) between the initiator and the memory, we plotted the DPU energy overhead as a surface dependent on these two parameters. The DPUs energy overhead for *Arch1* and *Arch2* is up to 4.5% and 7.5% respectively.

Increasing the packet length, the energy to transmit and translate the packet due to NoC routers and NIs increases proportionally. For the DPU components, energy does not increase because the lookup is done by the header independently of the payload length. As a results, the DPU energy overhead decreases with the increment of the packet length. The same behavior can be noted by varying the number of hops: the increment of the number of hops does not influence the DPU energy cost since the lookup is done on the NI. The increment of the network distance increases the energy



(a) Arch1 (b) Arch2

Figure 5.14. DPU energy overhead with respect to the energy consumed by the NoC subsystem for a memory access.

due to the transmission of the packet (NoC router) but not the energy due to the data protection (DPU).

5.5 Monitoring NoCs for Security Purposes

The protection mechanism described in previous sections represents an effective method for providing a secure access to memory locations, as well as secure transactions. However, while the presented protection mechanisms can help limit unauthorized accesses to protected memory blocks, attempts to illegal access in memory must be necessarily monitored to make the system aware of potentially dangerous behaviors. In fact, a compromised core executing malicious code can be used to perform a Denial-of-Service (DoS) attack against the system, with the aim of reducing system performance and operative life of battery and device. Moreover, as done for instance in modern anti-virus programs, sequences of accesses to memory and peripherals should be monitored to discover patterns that could imply the presence of an on-going attack to the system [19, 17].

In the following sections, we extend the security mechanism based on the use of Data Protection Units by defining a security monitoring system for NoC based architectures. Aim of the proposed system is to detect general security violations in the device and to allow an efficient counteraction for attacks addressing the communication subsystem and aiming at causing disruption or retrieving sensitive information. We focus in particular on the implementation of the basic blocks composing the security monitoring system. We detail the overhead associated with their implementation, and types of attack detected by the system.

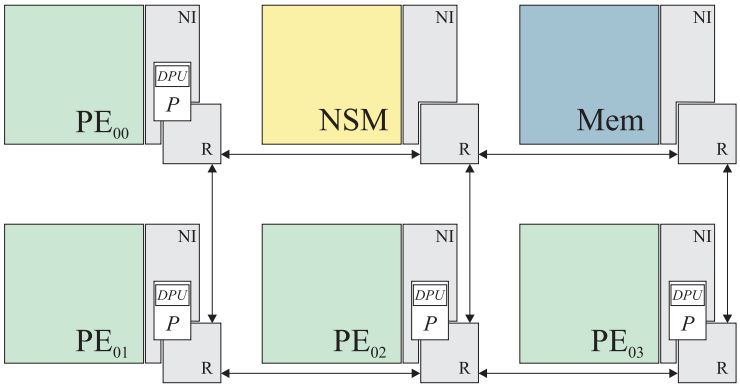


Figure 5.15. General NoC-based architecture including the security monitoring system.

Figure 5.15 shows a general NoC-based architecture including the monitoring system for detecting security violations. The monitoring system is mainly composed of three elements: *probes* (P), the *Network Security Manager* (NSM), and the *communication infrastructure* (NIs and Rs). Probes, collecting information about the NoC traffic, are implemented inside OCP compliant network interfaces [8]. In fact, as also previously explained, NIs represent the ideal position where to perform analysis of incoming traffic and discard malicious requests. The choice of embedding such activities inside NIs presents several advantages:

- Traffic is analysed when inserted by the core, therefore there is not need of *sniffing* packets to retrieve the information necessary for threats detection. As a consequence, the logic to implement probes is less complex and expensive in terms of area and power consumption.
- Monitoring is performed in parallel with operations performed by the NI kernel for the protocol translation from the OCP interface to the NoC. No overhead in performance is therefore associated to traffic analysis.
- Being probes embedded in NIs, traffic detected as malicious can be stopped or limited, and the relative core considered as compromised by the system. This allows a easier identification of the source of security violations and, if necessary, the "quarantine" of the core by sealing the NI.

The NSM, a dedicated core already introduced in section 5.3, takes care of collecting events and information coming from the several probes distributed in the system, analyzes the data received, and counteracts efficiently to the detected attacks. While focusing on hardware characteristics of the monitoring system, we leave to software designers the task of implementing detection strategies for security violations and appropriate countermeasures.

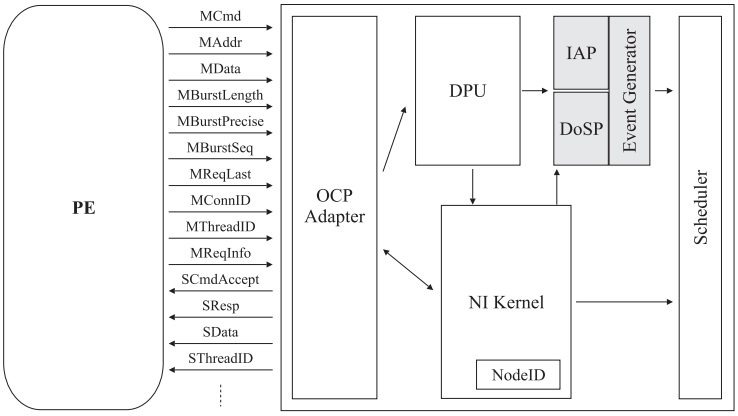


Figure 5.16. Architecture of the NI including the proposed probes.

Traffic produced by probes is to be kept divided from standard communication traffic inside the NoC. Although less impacting on the overall network traffic than the one generate for debugging activities [189], a higher priority must be given to this type of communication, in order not to be sensitive to denial-of-service attacks addressing NoC performance. Moreover, differently from the case of on-chip debug, messages coming from probes should not be accessible by external entities through public interfaces, in order to avoid the exploitation of the information collected to attacks the system.

5.6 Security monitoring system components

In this section, we give first an overview of the probes within the NI, discussing some of the concepts needed to understand their operation. Probes rely on the presence of DPUs embedded within the initiators' NIs. Moreover, we discuss the concept of *Event*, useful to describe communications with the central unit. In the second part of the section, we provide implementation details for the probes.

Figure 5.16 shows a NI embedding the probes we propose (in grey in the figure). The *Illegal Access Probe* (IAP in the figure 5.16) detects attempts to illegally access restricted memory blocks or range of addresses in shared memory systems, while the *Denial of Service Probe* (DoSP in the figure) is employed to detect *bandwidth reduction* or *draining attacks*. The *Event Generator* is triggered by the two probes and generates the packets to be sent to a central unit to communicate the security violations.

5.6.1 Events

Every probe generates events to notify the central unit security violations. As definition of event, we comply to what discussed in [129]: an event can be represented as

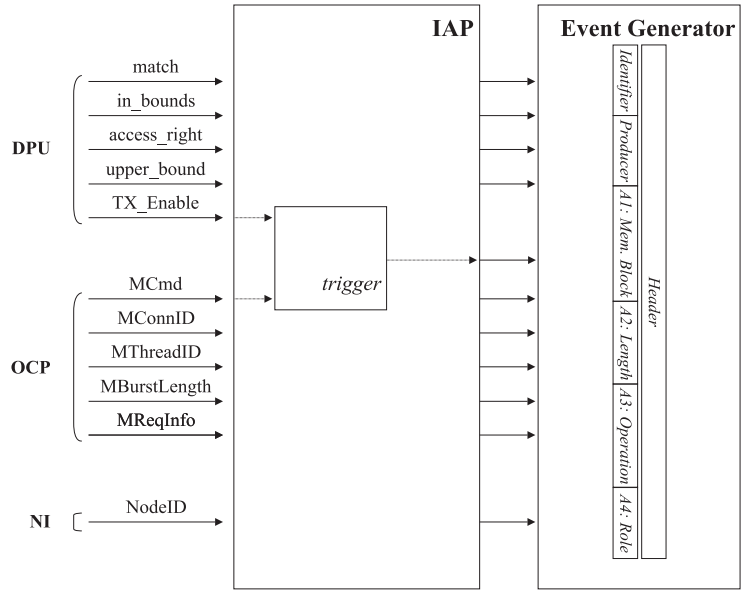


Figure 5.17. Illegal Access Probe: details.

a tuple composed by an *Identifier*, a *Timestamp*, a *Producer*, and several *Attributes*.The *Identifier* identifies events of a certain class of events, and it is unique for each class. The *Timestamp* defined the time at which the event was generated by the producer, identified by the field *Producer*. Attributes are represented in the form $Attribute = (AttributeIdentifier, Value)$ and the type of attributes and their values depend on the type of the event generated. In our case, *Identifier* specifies the type of symptom of attack detected by the probe, while *Producer* the combination of node, processing element, thread causing the illegal action. While not using *Timestamp* (we assume service packets transmitted in order and with prioritized or predictable latencies - statistics about timing are therefore generated inside the central unit), type of attributes are different for the two cases presented. Events generated are discussed in detail in next subsections.

5.6.2 Illegal Access Probe

As previously discussed, a data protection mechanism does not provide protection against the attacks described in section 3. Moreover, attempts to access unauthorized addresses should be notified to counteract efficiently the related security violations. In fact, an access to a non-allowed memory location could be due to software coding errors, as well as to the presence of a core compromised by buffer overflow or a draining attack.

Figure 5.17 presents architectural details of the *Illegal Access Probe* (IAP). The IAP is in charge of detecting the presence of attempts of unauthorized access to memory

locations and to notify the reasons of the alert to the central unit. The IAP triggers the creation of a packet to notify the security alert event to the central unit, passing the necessary information to the *Event Generator*. The event is generated when a new transaction is requested to the NI (*MCmd*) and the transmission of the packet is not enabled by the DPU (*TX_enable* = '0'). The IAP module takes as input OCP signals used for identifying the producer of the event and the attributes correlated, as well as some DPU signals, used for identifying the type of security alert. Three main types of event can be identified:

- *Entry input not present in DPU*: this case corresponds to a request in which the combination of the PE identifier and thread ID is not present among the entry lines of the LUT of the DPU, as well as the memory address targeted. This event is identified by a DPU's *match* line equal to '0'.
- *Out of block boundaries*: in this event, the request addresses input combinations recorded in the LUT of the DPU, while the length of the data to be stored or read would exceed the memory block boundaries. This event is detected when the signal *in_bounds*, which is true when *upper_bound* is higher than the sum of *MBurstLength* and the targeted memory address, is equal to '0'.
- *Wrong access rights*: while the previous two cases are satisfied, the access rights recorded in the RAM of the DPU for the input combinations are negative. The signal *access_right* is therefore equal to '0'.

The right part of figure 5.17 shows also the packet generated to communicate the event. The header of the packet depends on the specific NoC implementation and it is not discussed. As previously said, the event packet is composed of several fields. *Identifier* identifies the type of security alerts detected by the IAP. *Producer* is generated by the combination of the identifier - contained in the NI - of the node in the NoC (*NodeID*), the PE identifier (*MConnID*), and the thread identifier (*MThreadID*). Attributes sent to the central unit and relevant for analyzing the security alert are composed of the information of the unauthorized transaction, i.e., block of targeted addresses (given by the DPU's *upper_bound* signal), length of the data (*MBurstLength*), type of operation requested (given by the OCP signal *MCmd*), and role of the initiator (given by the OCP signal *MReqInfo*).

5.6.3 Denial of Service Probe

Access control solutions allow to stop unauthorized operations on restricted blocks of memory or ranges of addresses. However, they do not provide protections against attacks aiming at creating Denial-of-Service in the system for instance through the injection of useless packets. As discussed in section 3, these attacks can be carried out in mobile and multimedia systems with the goal of reducing resources bandwidth or

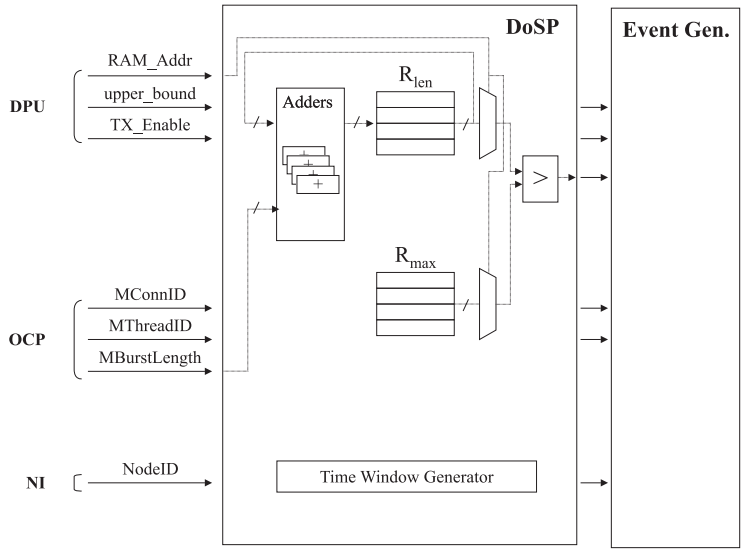


Figure 5.18. Architecture details of the DoSP.

battery life. In order to avoid such types of attacks, our monitoring system collects statistics about traffic inserted by every element active on the NoC. This would allow discovering unnatural behaviours performed in the system, as done by IDSs in data networks [181]. Goals of the *Denial of Service Probe* (DoSP) presented in this section are to collect information about the traffic generated by the processing elements interfaced to the NI and to notify the central unit of unexpected traffic conditions, interpreted as symptom of DoS attacks.

In this work, we consider as unnatural traffic conditions all deviations from the average bandwidth expected at design time. We monitor the bandwidth considering the data loaded / stored by an initiator from/to a specific memory block or range of addresses. The bandwidth - representing our statistic event - is calculated over a defined time window. The length of the time window is set by the central unit at run-time or by the designer at design time and depends on the selected security level.

Considering a generic discrete probability distribution for traffic profiles produced by a PE, with media μ and standard deviation σ, we consider unnatural traffic the one exceeding $\mu \pm m\sigma$, with m set accordingly with a desired security level. If those limits for a specific connection are exceeded, an event is generated and sent to the central unit.

Figure 5.18 shows architectural details of the *DoSP*. The probes triggers the *Event Generator*, as done already by the IAP. In this case, the DoSP monitors the amount of traffic to/from a selected memory space caused by an initiator (thread running on a PE). With respect to a complete monitoring of the lower and upper bounds of the traffic distribution, and to the monitoring of all the input combinations of the DPU, in the architecture shown in figure 5.18 we monitor a limited number of combinations. In

particular, we consider only transactions initiated by initiators acting as *user*. Moreover, we consider only the case of traffic exceeding the upper limit of the distribution ($\mu + m\sigma$). Therefore, without loss of generality, we assume under monitoring the entries combinations recorded in the first l lines of the protection unit, with $l < n$, where n is the total number of entry lines of the DPU. We believe this choice a good trade-off between the security service offered and the overhead of the implementation. We implemented the generation of the time window using a programmable counter.

When an initiator loads or stores data into an address in the monitored block i, the length of the data is added to the register $R_{len,i}$. The register is selected by the signals driving the RAM of the DPU (*RAM_Addr*).The new value stored in $R_{len,i}$ is compared to the maximum value allowed for the selected block in the current time window, stored in register $R_{max,i}$. In the case that the new value in register $R_{len,i}$ is higher than what allowed, an event is triggered and a packet is created to communicate the security alert. As with the IAP, the packet generated to communicate the security alert event to the central unit is composed of the *Identifier*, which identifies the type of security alerts detected by the DoSP, the *Producer*, generated by the same signals creating the IAP *Producer*'s field, and *Attributes*, in this case containing information about the memory block addressed by the DoS attacks (DPU's *upper_bound* signal). Once the time window reaches its end, the value stored in register $R_{len,i}$ is reset, and statistics begins are collected for the following time window.

With reference to the full coverage of possible entry configurations, it is easy to see how the hardware blocks shown in figure 5.18 should be replicated for every entry line of the DPU. Monitoring the lower level of the bandwidth implies instead waiting for the end of the time window to avoid false alerts. Moreover, in the case of monitoring of traffic incoming from initiators with *superuser* and *user* roles, hardware blocks now described should be duplicated.

5.6.4 NSM and communication infrastructure

Goals of the NSM in the security monitoring system are to collect security alerts coming from probes and to elaborate appropriate countermeasures to attacks and problems detected. For these purposes, the NSM can be implemented in ASIC as a dedicated core, as a general purpose processor or as mixed implementation. However, software (or reprogrammable logic based) implementations allow a higher degree of flexibility, necessary to adapt and update the system in order to be able to face threats coming from new malware. In our architecture, we opted for this solution.

Another point to be considered in the implementation of the secure monitoring system is the communication infrastructure. As already mentioned, the traffic coming from probes should be kept separate (at least virtually) from traffic coming from initiators, in order to avoid DoS attacks to influence security service communication. As reported in [189], three main options can be considered: *Separate Physical Interconnect for the original NoC application and the NoC Monitoring Service*, *Common Physical*

Table 5.4. Area and energy consumption of the elements composing the security monitoring system.

	Area $[\mu m^2]$	**Energy** $[pJ]$
IAP	56.48	0.088
DoSP	25699.38	30.820
Event Gen.	968.26	1.123
DPU	600041.96	72.970

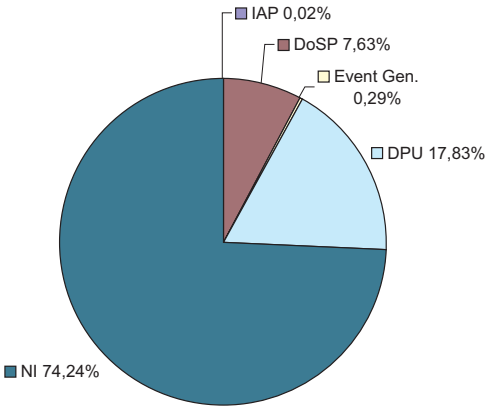


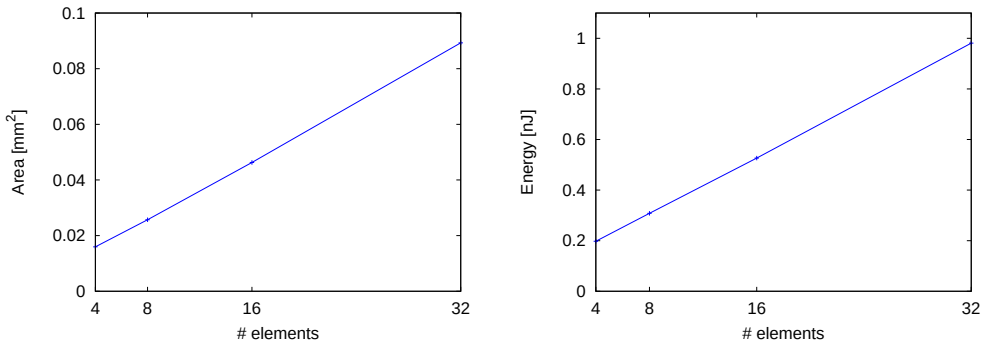
Figure 5.19. Area breakdown of the several elements of the security monitoring system inside the NI.

Interconnect but Separate Physical NoC Resources, *Common Physical Interconnect and Shared Physical NoC Resources*.

We have chosen in our case to consider the third option, i.e, sharing all the NoC resources while keeping the NoC user traffic and the monitoring traffic separated, therefore creating a virtual NoC for monitoring. This solution is particularly convenient in our case, being the monitoring traffic not relevant and the overhead associated to the NoC implementation limited.

5.7 Probes Synthesis results

In this section, we present synthesis results for the implementations of the probes presented in Section 5.6, obtained by using the 0.13μm HCMOS9GPHS STMicroelectronics technology library. In table 6.1 we show area (in μm^2) and energy consumed (pJ) of the proposed components, i.e., the IAP, DoSP and *Event Generator*. Values for a DPU with 16 entry lines are also shown for comparison. The value for the DoSP refers



(a) Area (b) Energy

Figure 5.20. Synthesis results of the DoSP by varying the number of elements monitored.

to an implementation monitoring 8 input combinations. The synthesis was optimized for a clock frequency of 500 MHz.

Figure 5.19 shows the area breakdown (in mm^2) for a NI including a DPU module and the two probes. The DPU has 16 entries, while the DoSP monitors 8 input configurations. The area occupied by the IAP is around the 0.02% of the overall NI [9], impacting therefore not significantly to the overall area budget. This is mainly due to the fact that the IAP is mainly composed of combinatorial circuits, reacting to changes of the input signals to provide a trigger to the *Event Generator*. A bigger impact is given by the area consumed by the DoSP (7.63% for 8 configurations monitored). The overall security system, including the DPU and the two probes, counts for around 25.6% of the NI implementation.

In figure 5.20(a) we show the area occupied by several DoSP implementations, by varying the number of input configurations monitored (4, 8, 16, 32). As expected, the area increases linearly with the number of monitoring blocks, increasing in fact the number of registers used for storing statistics about traffic and maximum values allowed. Similar trends can be also observed for the energy consumed by the component (shown in figure 5.20(b)).

5.8 Summary

This chapter presented a secure communication system conceived for dealing with the weaknesses intrinsic of a multiprocessor system with shared memory and for exploiting the characteristics of the Network-on-Chip to detect and prevent them. In particular, we presented an innovative solution for data protection in Multiprocessor System on-Chip architectures based on NoC. The solution is based on a hardware module (called *Data Protection Unit*) that is integrated into the network interface and that guarantees

secure accesses to memories and memory-mapped peripherals. The proposed solution takes into consideration a distributed infrastructure. We studied different design alternatives applicable to the target distributed shared memory architecture, and the design of a central unit that dynamically manages the data protection modules proposed in the chapter. The experimental results obtained by considering two different case studies have shown that the introduction of the DPU has only a limited area and energy overhead (up to 17% and 7.5% respectively) without impacting the system performance.

Moreover, we presented a monitoring system for NoC based architectures based on the proposed data protection units, with the goal of detecting security violations carried out against the system. Information collected are provided to the central unit for efficiently counteracting actions performed by attackers. We detail architectural implementation of two type of hardware probes, i.e., the *Illegal Access Probe*, in charge of detecting the presence of attempts of unauthorized access to memory locations, and the *Denial-of-Service Probe*, which detects unnatural traffic behaviors. We analyzed the overhead associated with an ASIC implementation of the monitoring system, discussing type of security threats that it can help detect and counteract.

The work presented in this chapter resulted in publication in several conferences [17, 190, 191], a book chapter [192], a journal paper [54], and two patent applications [193, 194].

Chapter 6

Networks-on-Chip Monitoring

The classical approach for observing system performances and behavior involves the use of a monitoring subsystem for detecting, collecting, and interpreting run-time information collected during the system execution. Monitoring of system resources and behavior is needed for testing, debugging, performance optimization, as well as for run-time tuning of resources utilization. This chapter deals with the aspect of NoC monitoring, and presents the work performed for implementing a monitoring system for NoCs, as well as discussing costs and overhead associated with to its utilization. The approach proposed in this dissertation represents an attempt to provide to application and system designers with a comprehensive study on the set of tools and information that can be exploited at run-time or during the post-manufacturing phase for the optimization of an NoC architecture.

The remainder of this chapter is organized as follows. Section 6.1 discusses contributions of this dissertation with respect to the state of the art. Section 6.2 presents an overview of the proposed NoC monitoring architecture. Section 6.3 discusses implementation details of probes deployed in the system, while section 6.4 presents the strategies adopted for data collection and storage of detected information. Section 6.5 discusses the experimental results obtained when employing the monitoring system at design time for profiling a *ray tracing* application (subsection 6.5.1), and at run-time for optimizing the NoC operating frequency while running an audio-video multimedia system (subsection 6.5.2).

6.1 Contributions with respect to the state of the art

Compared to previous work, we address the problem of NoC monitoring from the point of view of the architecture to be deployed on the system, while focusing on the different design trade-offs related to its implementation. Compared to [129] and [130], the system we propose focuses on the automatic collection of data without altering the execution of the events on the NoC, and without the need of employing de-packing of

the messages, being events collection performed at the NI. As in [133], we suggest the use of the monitoring system for dynamic adaptation. However, in our work we propose a solution that provides the designer with a larger set of information other than the link utilization, allowing a more detailed view about the system behavior. Moreover, we propose a set of preprocessing operations that can significantly reduce the amount of traffic generated by the probes, therefore limiting the intrusiveness of the monitoring activities. With respect to previous work on debugging and testing, the implementation of our monitoring system results in lower complexity and overhead, due to the fact that we focus on monitoring specific operations of the NoC, and not on the observation of the circuit signals and registers' values. We acknowledge however the fact that more complex debugging systems such as the ones presented in [129, 127] and [128] can be reused for instance for monitoring at run-time the system behavior, provided additional support for the run-time management of the received data is available. In [44, 53], data detected are collected by activating an Interrupt Service Routine which influences the program execution and interferes with the system behavior. On the contrary, in our work we detail a solution in which data collection is initiated by the probes: such solution does not influence processing cores execution, and it is more suitable for a distributed and heterogeneous system such as an NoC-based MPSoC. The industrial case in [137, 138] presents several similarities with the solution discussed in this dissertation: more specifically the run-time programming the probes, as well as the possibility to automatically send the monitoring results to a central management unit, are concepts that have been addressed and implemented, even though with different trade-offs in costs and flexibility, both in [137] and in our solution. In this dissertation, however, we propose a probe architecture which provides the designer with better flexibility on the choice of the amount of on-chip resources to devote to the monitoring, considering that the *Statistics collector* implemented in [137] has a granularity of 8 probes (i.e., each *Statistics collector* can provide from 1 to up 8 input channels for the probes). In our solution, the number of probes that can be instantiated in one tile is not restricted to any configuration. Moreover, in our work, we target a wider and more detailed range of events, as well as providing a set of pre-processing functionalities which can reduce significantly the amount of bandwidth needed for the communication of the monitoring data.

The main contributions of this work can be summarized as follow:

- We perform a comprehensive study of the most common system events that can be object of detection in the communication subsystem, including those events about cores and processing units that can be observed, with limited intrusiveness, from the NoC monitoring architecture;

- We introduce a multi-purpose programmable probe, located at the network interfaces of the NoC, that can provide useful information about core activities by analyzing transactions and requests passing through the communication chan-

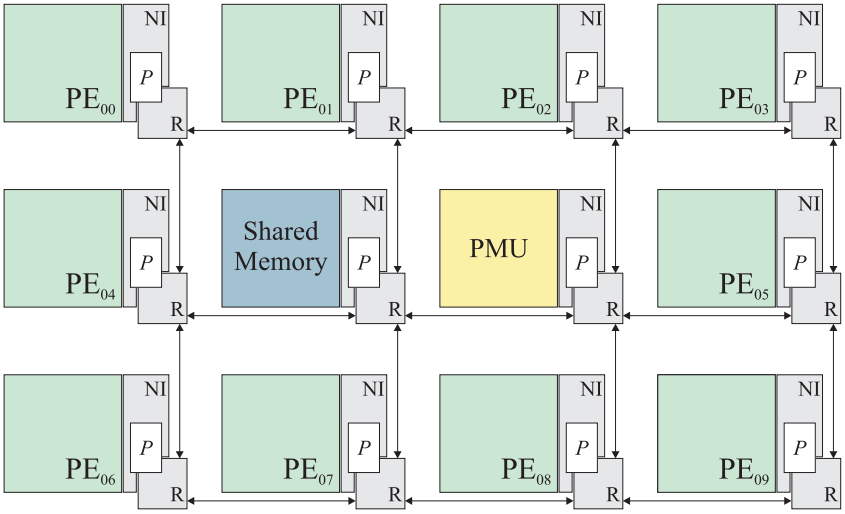


Figure 6.1. Overview of the NoC monitoring architecture.

nel. Probes in NIs do not influence core operations, and a relatively small overhead is paid in terms of area and energy consumed by the monitoring system, as well as in the amount of monitoring traffic data generated by probes.

- We propose and discuss an architecture for the efficient and automatic collection and storage of the information related to the detected events, and we evaluate the intrusiveness of the monitoring system. Data monitoring and their early processing is performed by NIs in parallel to protocol translation, thus limiting the intrusiveness and the impact of the monitoring architecture on system performance.

- We propose a strategy for managing collected information at system level, for improving performances and management of system resources at run-time.

6.2 Overview of monitoring Architecture

As reference platform, we consider the same shared-memory multiprocessor architecture introduced in chapter 3. Figure 6.1 shows a generic NoC architecture including the processing elements (PEs), the shared memories, and the four main components of the monitoring system discussed in this dissertation, i.e.:

- *Probes* (*P* in figure 6.1): we located probes inside each NI. By snooping OCP signals at the interface of the core, probes observe cores' operations and events. The probes can also monitor resources utilization and the generation of communication events by detecting signals behavior in the router and the NI.

- *Probes Management Unit* (*PMU*): a centralized element is in charge of the configuration of the probes, of the retrieval of the collected data, and of its elaboration both after the execution of applications and at run-time, depending on the functionalities offered to the platform.

- *Data collection subsystem*: data generated by probes are collected in order to be stored and/or processed by the PMU. Traffic generated by probes have to be taken into account in the design of the interconnection system.

- *Data storage subsystem*: data collected and transmitted by the probes are stored locally in registers or in a local memory of the PMU for being used at run-time, or in a on-chip streaming memory for being processed after execution (not shown in figure 6.1).

The four generic components will be presented in detail in next sections, by providing a description of their functionalities and of the respective design trade-offs.

6.3 Programmable Probes

Probes are in charge of detecting events generated by the system, as well as of initiating the procedure for data collection. Probes are embedded within the NI of each tile, at the boundary between the NoC and the core. They observe signals of the OCP interface, as well as internal signals of the NI and the router. For our monitoring system, we propose the implementation of a multipurpose probe that can be programmed for detecting several types of event.

Figure 6.2 shows the general architecture of our multipurpose programmable probe. It is composed of six main elements:

- An *Events Detector*, that reacts every time the event to be monitored occurs.

- An *Accumulator*, used for collecting measured values for the observed event.

- A collection of *preprocessing modules* for early data elaboration.

- *Configuration Registers*, used to configure the probe for monitoring different events.

- A *Message Generator*, which creates messages to be sent to the PMU and the storage elements, and inserts them into the probe output queue.

- An *output queue* for buffering the message generated by the *Message Generator* before being scheduled for transmission.

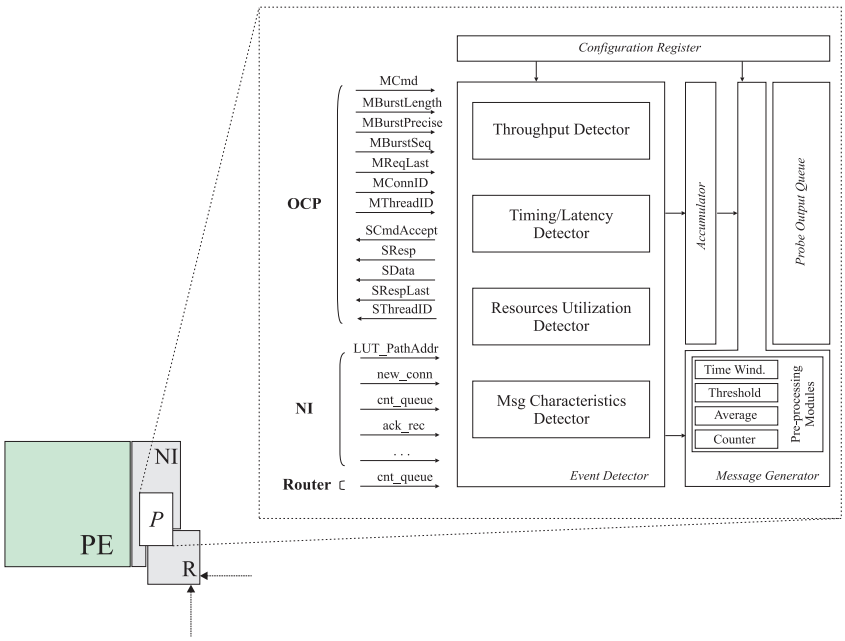


Figure 6.2. Architecture of the programmable probe.

6.3.1 Events detectors

The event detector observes OCP signals as well as NI and router internal signals. While in [44] probes are specialized for single events, we have chosen to implement a multipurpose probe that can detect all the types of events described in chapter 3 by selecting the event to be monitored in the probe configuration registers. As it will be shown in section 6.3.5, with this choice we trade-off the number of events simultaneously detectable with a strong reduction in the area overhead, while still keeping the possibility of detecting all the events. This choice allows designers to decide the amount of space in each tile and on the chip that will be spent for the monitoring system, by deciding the number of multipurpose probes to be deployed in each tile and, therefore, the number of events that it is possible to monitor in parallel in a single execution of the application.

Event detectors operate in parallel with the operations performed by the NI kernel. In this way, event detectors do not interfere with the operations of address translation or packing/de-packing performed by the NI kernel, thus satisfying the requirements of non intrusiveness needed for avoiding the *probe effect* previously described in chapter 3. Event detectors detect changes in the signals at the interface, without adding latency to the system. Following the classification described in chapter 3, event detectors can monitor the following types of events: *throughput events*, *timing/latency events*, *resources' utilization events*, and *message characteristics and statistics* about messages and packets generation. In the following subsections, we describe the implementation

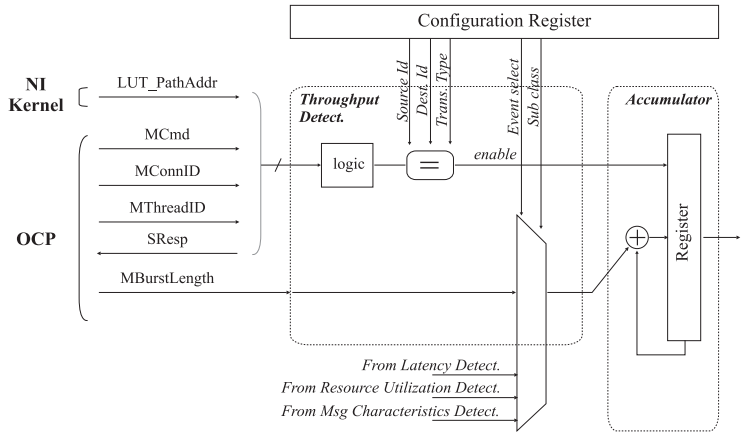


Figure 6.3. Throughput Detector.

of the event detectors, as well as the modules involved in the preprocessing of the collected data, and in the generation of the messages to the PMU. We moreover discuss the configuration and the programming model for the probes.

Throughput Detector

Throughput is evaluated by keeping track of the amount of traffic to/from a selected range of addresses generated by an initiator (i.e., a thread running on a processing element). Figure 6.3 shows a more detailed view of the *Throughput Detector* module. The connection to be monitored by the probe is written in the probe configuration registers. When an initiator begins a transaction, it drives the *MCmd* signal of the OCP interface of the NI, by specifying the type of operation (load/store) to be performed on the target. When the elements identifying the transaction (i.e., initiator, target, type of operation) match those specified in the configuration registers, the accumulator is enabled and the information about the amount of data transferred in the transaction (carried by the OCP signal *MBurstLength*) is added to the accumulator. The OCP signals involved in the detection are *MCmd*, which specifies the type of operation (load/store, i.e., incoming/outgoing traffic), *MConnID* (connection identifier) and *MthreadID* (thread identifier), which identifies the initiator, and signals coming from the look up table (LUT) of the NI Kernel (*LUT_PathAddr*)[195]. *LUT_PathAddr* drives the RAM of the LUT looking up the target memory address present as input of the OCP *MAddr* signal to retrieve the routing information to be inserted in the header of the packet [10]. Throughput can be collected for the whole execution, for specific time windows, and for the different connections active at the initiator.

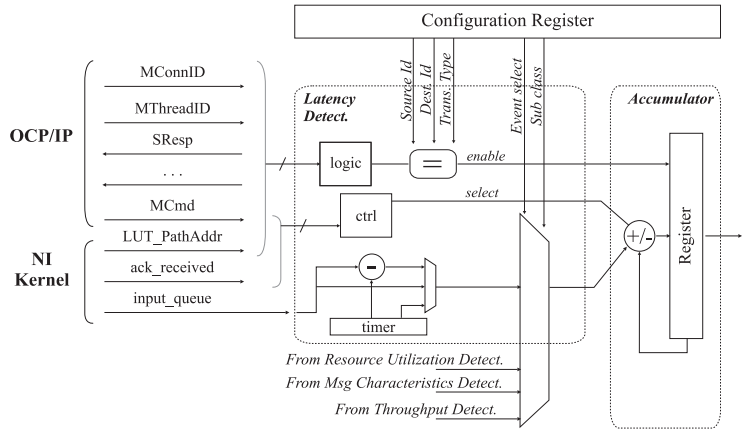


Figure 6.4. Timing/Latency Detector.

Timing/Latency Detector

The *Timing/Latency* module is in charge of measuring time properties of transactions. Details about its implementation are shown in figure 6.4. It is able to measure the latency between:

- An initiator's request and the reception of the acknowledge message (initiator-to-initiator transaction (*I2I*)).
- The time needed for a request (initiator-to-target transaction (*I2T*)).
- The time needed by the target for executing its task (execution time (*EXEC*)).
- The time needed by the acknowledgement message of the target to reach the initiator and complete the transaction (target-to-initiator (*T2I*)).

When the initiator begins a transaction, the probe detects a change in the OCP signal *MCmd*. Depending on the latency measurement, the probe:

- Stores a timestamp in the accumulator which is subtracted from the arrival time when the acknowledgement message is received (*I2I* transaction).
- Stores in the accumulator a timestamp and subtracts it from the arrival time of the initiator request detected at the target - the probe sends along with the initiator message a communication for the target to timestamp the arrival time of the initiator request, and to send back this information along with the acknowledgement (*I2T* transaction).
- Communicates to the target that it should measure execution time and to send the information along with the acknowledgement, which is stored in the accumulator (*EXEC*).

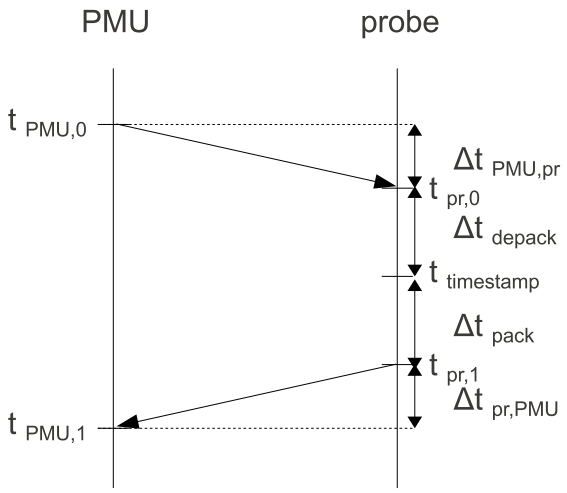


Figure 6.5. Synchronization protocol between PMU and probes.

- Communicates to the target that it should send timestamp of acknowledgement message generation, to be subtracted from arrival time at the initiator, and stored in the accumulator (*T2I* transaction).

Latency can be collected for every single transaction, and for different connections, monitoring the same signals and using the same procedure described for the *Throughput Detector* module for selecting the desired connection.

Timers synchronization

The use of the *Timing/Latency* event detector requires the collaborative action of probes at the initiator (shown in figure 6.4) and at the target of the transaction. Therefore, before executing timing/latency measurements, the system should synchronize timers at initiator and target. However, timing synchronization in a distributed architecture may represent a challenge, due to the problem of being able to synchronously start or tune timers in two different areas of the chip. In fact, in large designs, signals may take several clock cycles to be transmitted from one area of the chip to another.

In [129], timestamping of messages (and events ordering) is supported by a totally synchronous NoC. However, for a large NoC a fully synchronous behavior is not likely to be implemented due to signal delay reasons, and the timestamping methodology proposed in [129] would not be able to work without some sort of time synchronization between the tiles of the NoC. Abstracting the synchronization at the transaction level [196] would not equally help in our goal, i.e, being able to calculate precise timing information when two or more actors are involved in the transaction.

We therefore developed a synchronization methodology inspired by a simplified version of the Network Time Protocol (NTP), the current widely accepted standard for

synchronizing clocks over the internet [197]. The synchronization protocol is shown in figure 6.5. As basic assumption, we consider having in NoC tiles counter-timers running at the same clock frequency. For each node in which timestamping is needed, at time $t_{PMU,0}$ the PMU sends a *read* request to the timer register of the node. $t_{PMU,0}$ is read from the PMU local timer and stored. At time $t_{pr,0}$, the message is received by the probe and, after Δt_{depack} cycles in which the received message is processed, the value of the probe timer is read ($t_{timestamp}$) and inserted in the acknowledgement message sent back to the PMU at time $t_{pr,1}$, after the packet has been created. The acknowledgement message is received by the PMU at time $t_{PMU,1}$.

The roundtrip delay d can be calculated as:

$$d = (t_{PMU,1} - t_{PMU,0}) - (t_{pr,0} - t_{pr,1})$$

while the timer offset t_{offset} between the PMU and the probe can be defined as:

$$t_{offset} = \frac{(t_{PMU,0} + t_{PMU,1}) - (t_{pr,0} + t_{pr,1}) + (\Delta t_{PMU,pr} - t_{\Delta pr,PMU})}{2}$$

where $\Delta t_{PMU,pr}$ is the time needed for the message to travel from the PMU to the probe, while $\Delta t_{pr,PMU}$ is the time needed for transferring a packet in the opposite direction. By assuming at synchronization time that the latency of transmitting packets from PMU and probe and vice versa is constant, and $\Delta t_{PMU,pr}$ equal to $\Delta t_{pr,PMU}$ (as it is possible to observe for instance in regular topologies such as meshes), t_{offset} can be expressed as:

$$t_{offset} = \frac{(t_{PMU,0} + t_{PMU,1})}{2} - t_{timestamp} + \alpha$$

where α is a value that can be considered constant and that depends on the difference between Δt_{pack} and Δt_{depack}. Once evaluated by the PMU, the value of t_{offset} is stored in a register in each probe, and used in the calculation of the timestamps. Synchronization can also be repeated at run-time, in the case, for instance, of the reconfiguration of the probes, or for retuning timers offsets due to mismatches in counters clock frequencies.

Resources' Utilization Detector

At network level, the measurement of *resources' utilization* is performed by monitoring the status and the occupation of the internal queues of NIs and the routers. In figure 6.2, *cnt_queue* represents the signal monitored for this purpose by the resources' utilization event detector. These signals are directly taken from the counter used in general implementations of a hardware FIFO for measuring the number of slots occupied, and for calculating whether the FIFO is full or not [198]. From the configuration registers it is possible to specify which FIFO, among the NI's and router's ones, should

be monitored, and select the right counter signals to sample. The information about the utilization of the resources is sampled at intervals set in the configuration registers, and generated by using a programmable counter internal to the probe. For every sample, the information detected is stored in the accumulator. A message is therefore generated by the message generator and sent to the PMU. The *Resources' utilization Detector* can also be programmed to generate an event only if the monitored FIFO is full.

Messages' Characteristics Detector

The event detector monitors also characteristics of message generated, and general NoC characteristics. In particular, the *Messages' Characteristics Detector* aims at detecting *user configuration events* and *NoC configuration events* [129]. *User configuration events* include the detection of information about the characteristics of the communication between two cores, such as the connection identifier, the type of connection, the amount of data transferred, and so on. *NoC configuration events* are mainly related to communicate modifications in the configuration of the elements of the NoC (such as changes in the entries of a routing table in a router), and of the system in general. Other types of more specific events detected by the *Messages Characteristics Detector* include also information about the number of transactions generated by an initiator, the type, length and connection characteristics of messages between cores, and type, length and connection characteristics of packets created. These measurements are useful only in those NoC architectures in which a certain degree of dynamic adaptation on the communication characteristics is allowed by the system.

6.3.2 Data preprocessing

The detection of some of the events can generate traffic that could be too expensive to be supported or managed. In order to reduce the traffic generated by the probes, we implemented the possibility of pre-processing collected data before sending them to the PMU. Pre-processing is enabled by selecting the desired functionality in the probe configuration registers. The following pre-processing functionalities were implemented:

- **Time windows:** total execution time can be divided in time windows, and messages are generated at the end of each of them. The maximum number of sent messages is therefore reduced to the number of time windows, instead of sending messages for every occurrence of an event. Time windows are generated through the use of a 32 bits programmable counter. The length of a time window is set in the configuration registers;

- **Threshold:** by allowing this feature, messages are generated only after comparing the event measured and stored in the accumulator with a threshold specified

in the configuration register. A message with the collected information is generated if the data is higher, equal, or lower than the specified threshold. In this way, only critical information is sent, and, as shown in the section 6.5, transmitted data can be reduced significantly;

- **Average:** traffic generated by individual samples can be reduced by averaging the collected values over the number of collected samples in the execution or in the time window. At every event detection, the measured value is added to the accumulator, while increasing by one an internal counter. At the end of the averaging time, only a message containing the sum of all the collected values and the number of occurrences is sent, delegating to the PMU the execution of the division between the two for the exact calculation of the average. This solution avoids the implementation in the probe of an expensive hardware divider, while still obtaining the reduction in the traffic generated.

6.3.3 Message Generator

The *Message Generator* creates packets to communicate to the PMU the event or set of events detected. We propose a data collection system in which the generation of monitoring messages is automatically triggered when reaching the end of a time frame, when a specific event occurs, or when reaching the end of the application. When one of these events occurs, the *Message Generator* generates a packet whose payload contains the information kept in the accumulator, leaving the accumulator ready for the following events. It acts as an *initiator* performing a *write* operation on a memory location, and it sends the packet to a specific register or memory address associated with the probe. Message packets are composed of a header and a number of payload flits (up to 3) that varies depending on the collected event. The header of the packet contains information about the source of the communication (the probe generating it) and the destination where to store the collected value. The destination is generated using the memory address stored in the *Storage_Address* field of the configuration registers, which contains information about the memory location associated with the probe. The PMU associates each probe with its identifier, its configuration, and the memory address where to store the collected data, in order to retrieve and recognize the data read during the elaboration. The packet generated is inserted in the probe queue (*Probe Output Queue* in figure 6.2), waiting to be scheduled for being transmitted.

For certain types of events in which data transmitted can be coded with a small number of bits (for instance the queue occupation), the message can contain information about more than one event, in order to reduce the bandwidth needed by the monitoring system. For instance, considering 4 bits for encoding the occupation of a NI input FIFO long 8 stages, with a data width of 32 bits, up to 8 samples can be grouped and sent together in one single message with 1 header and 1 payload flit, reducing

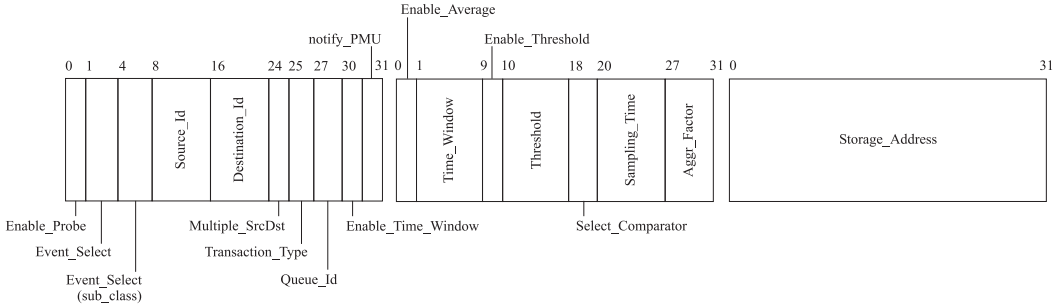


Figure 6.6. Probe Configuration Registers.

therefore the required monitoring throughput by around 87%. Using the maximum length event message (represented in our monitoring system by 4 flits packet, i.e., 1 header flit and 3 payload flits), throughput can be reduced up to 92%. Data aggregation is selected by enabling the appropriate bit in the probe configuration register (*Aggr_factor* in figure 6.6). The aggregation factor (α) can be set from to *no aggregation* to the *maximum aggregation*, in which packets, with 3 payload flits containing each the information of 8 events, are generated. The use of the data aggregation depends on the type of application to be monitored, on the type of event, and on the delay allowed between the detection of the event and its delivery to the PMU.

6.3.4 Probe configuration

Figure 6.6 shows the *Probe Configuration Registers*, and all the register fields that can be selected for enabling the functionalities of the probe. The *Event_Select* field is used for selecting the general class of event to be monitored, while *Event_Select (sub_class)* allows to specify the subclass of the event. As an example, if the latency is the event to be monitored, *Event_Select (sub_class)* can select one between the initiator-to-initiator transaction (*I2I*), the initiator-to-target transaction (*I2T*), the target execution time (*EXEC*), or the target-to-initiator transaction (*T2I*). *Source_Id* and *Destination_Id* specify the source and the destination identifier of the connection to be monitored by the probe, and these values are compared during the collection of events with the signals identifying a connection received by the OCP interface and the NI, i.e., *MConnID*, *MthreadID*, and *LUT_PathAddr*. The bits of the *Multiple_SrcDst* field specify the connections to monitor between initiators on the source core and targets on the destination core (*1-to-1*, *1-to-all*, *all-to-1*, *all-to-all*). *Transaction_Type* specifies the operation to be monitored by the probe, i.e., *read*, *write*, or *all*. *Enable_Time_Window* and *Enable_Threshold* enable the corresponding preprocessing features. *Time_Window* stores the value to be considered as the length of the time window, while *Threshold* the value to be used as comparison with the value collected by the accumulator of the event detector. *Select_Comparator* selects whether a *greater-than*, *equal*, or *less-than* operation

Table 6.1. Cost of probes and tile components.

	Area (mm^2)
Single multipurpose probe	0.042
Timing/latency probe	0.037
4 multipurpose probes	0.156
Router (5 p)	0.143
NI initiator	0.141
NI target	0.172
ARM920T	4.7

should be performed on the data compared with the threshold. *Enable_Average* enables the use of the average functionality. *Queue_Id* is employed to select the internal queue to be monitored by the probe, while *Sampling_Time* to set the time (in clock cycles) between two consecutive measurements of the number of elements contained in the buffer. The *Storage_address* field contains the memory address storing the value of the event detected by the probe. An additional bit in the configuration register (*notify_PMU*) is used for specifying whether notifying the PMU of the arrival of the event message from the probe, in particular in the case of the use of the monitoring system for the run-time management of the platform. In the case of *notify_PMU* enabled, messages transmitted by the probes generate an interrupt at their arrival at the NI of the PMU, allowing the central unit to immediately receive the information.

Configuration registers are memory-mapped, and, as explained in section 6.4.3, the PMU configures them before program execution or at run-time. The PMU keeps a record of the configuration of each probe, in order to be able to interpret correctly and process the data received, together with the information about the memory address or register in which the collected data will be stored.

6.3.5 Implementation cost

We implemented the monitoring probes in VHDL and synthesized them by using Synopsys Design Compiler, considering a 0.13μm technology library, and optimizing the synthesis for a clock frequency of 500 MHz.

Table 6.1 shows the area, in mm^2, of a single multipurpose probe, as well as the cost of a monitoring system composed of 4 probes, able therefore to monitor 4 different events in parallel in the tile. The implementation of a multipurpose probe implies an overhead in area of around the 13% with respect to the most expensive single monitor, i.e., the *timing/latency* probe. In the evaluation of the area occupied by the *Timing/latency* probe, both probe components at the initiator and at the target where considered. As it is possible to notice, the area used by a 4-probe system is slightly smaller than the area of a single probe multiplied by 4. This is due to the fact that some components of the multipurpose probe, such as the programmable counter used

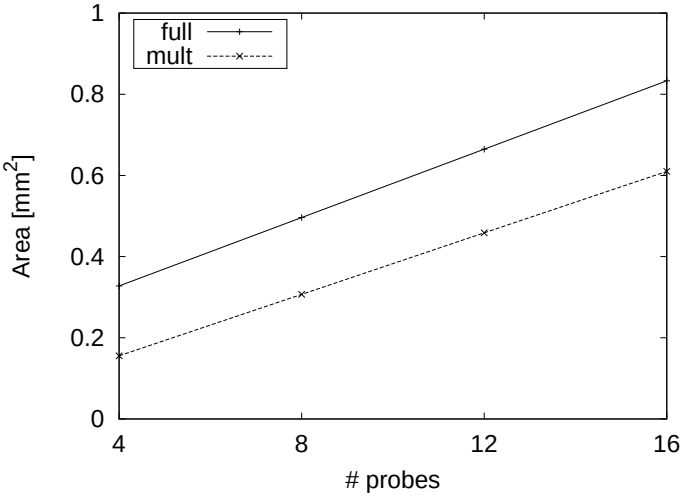


Figure 6.7. Area of a monitoring system composed of multipurpose probes (*mult*) and "full" probes (*full*), while varying the number of probes and the number of instances of the same event detectable in parallel.

for generating the sampling signal, are shared by all the events generators of the probe.

An alternative choice to the use of multipurpose probes could have been to implement all the event detectors separately, such as for instance in [44]. However, the cost of a complete implementation would have been prohibitive, in particular because of the high number of possible connections to be monitored, and of the significant amount of buffering space needed for accumulating locally at the probe all the possible events for guaranteeing non-intrusiveness in the message generation. The saving in area obtained when comparing a monitoring system with 4 multipurpose probes with such a "full" monitoring system able to monitor all the events in parallel is around 52%, being its estimated area of 0.328 mm^2. In the comparison, we considered the "full" monitoring system having probes for every type of detectable event, and able to follow 4 different connections at the time [44, 195]. Figure 6.7 compares the scalability of the two types of monitoring systems while varying the number of probes, and, therefore, the number of events detectable in parallel in the same tile. As is it possible to notice, the monitoring system implemented by using multipurpose probes allows a significant saving in area in all the configurations.

The 4-probe system counts for approximately the 35% on the total area of the NI and the router. Reference routers and NIs are those obtained when generating a system similar to the one shown in figure 6.1. The reference architecture represents a typical shared memory multiprocessor composed of 10 initiators and 1 target-memory. Tiles are positioned in a mesh topology. The architecture implements a table-based routing algorithm and wormhole control flow. Arbitration of output queue is based on Round Robin, among elements of the same class of messages (NoC data or probes messages).

Table 6.2. Energy per operation of the event detector components.

	Energy (pJ)
Throughput probe	0.29
Timing/latency probe	2.10
Resource utilization probe	0.15
Message characteristics probe	0.87
Accumulator	11.51
Output queue	12.90

Table 6.3. Energy dissipation due to the NoC components considering a 32-bit datapath running at 500MHz.

	NoC router			**NI initiator**	**NI target**
	3p	**4p**	**5p**		
Header Energy [pJ]	75.2	80.1	88.1	92.7	107.3
Payload Energy [pJ]	63.3	65.1	67.8	44.7	52.3

NIs and routers were obtained by using the PIRATE-NoC compiler [184]. Buffers length was imposed equal to 4 for the routers, while equal to 8 and 16 respectively for NI at the initiator and target.

Considering an NoC with a monitoring system with 4 probes in each tile and one without it, the overhead is 55% of the total area of router and NI in the tile. Overhead obtained is of the same order of magnitude of probes implemented for debugging purposes [129]. The overhead obtained when considering in the evaluation also the processing element in the tile is approximately the 3% (we consider as reference a typical embedded processor implemented with the same 0.13μm technology library, i.e., an ARM920T with 16KB of data and instruction caches) [41].

Table 6.2 shows energy values per operation associated with the data detection of each different event of the probe, as well as to the other components of the probe. The energy was obtained by using Synopsys Design Compiler and Prime Power with the same 0.13μm technology library. The energy consumption of each event detector is proportional to its architectural complexity and the amount of storage elements needed for performing its operation; the *Timing/latency* probe is for this reason the most power hungry event detector, due to the monitoring options available and to the protocol implemented for calculating the latency of the transactions. As a comparison, we also reported in table 6.3 the energy dissipation for the network components obtained by using the PIRATE-NoC Compiler [184].

Summarizing, looking at the synthesis results both energy and area overhead can be considered acceptable given the provided service at network level provided by the monitoring infrastructure.

Table 6.4. Traffic generated by a probe for notifying several detected events.

Event	Bandwidth
Throughput for execution	$\frac{64bits}{execution\_time}$
Throughput for time window	$\frac{64bits}{time\_window}$
I2I latency for transaction	$\frac{64bits*number\_transactions}{execution\_time}$
I2I latency above threshold	$\frac{64bits*number\_transactions}{execution\_time}$
queue utilization	$\frac{64bits}{queue\_sampling\_time*\alpha}$
Average I2I latency in time window	$\frac{96bits}{time\_window}$

6.4 Data management

In this section, we discuss data collection, storage, and management of messages generated by the probes. In a distributed system, such as an NoC-based architectures, the collection and storage of information detected represents a challenge for the designer. In [44] and [53], data collection is performed running an interrupt service routine that reads at regular intervals data detected by the probes. However, in [44], such a strategy stops normal core execution. Moreover, it is either not applicable in the case of probes distributed in the different tiles of the NoC, or, it would imply storing locally the information detected and requesting the PMU to read them from the local memories. On the one hand, this strategy would postpone the transmission of information at times decided by the central probes manager, increasing the time between the detection of the event and its processing by the PMU. On the other hand, it would increase the amount of traffic needed for obtaining the information from the probes, in particular in the cases in which warning messages (for instance, events over a specific threshold) are transmitted. As introduced in 6.3.3, in our approach, warning messages are automatically triggered when reaching the end of a time frame, when a specific event occurs, or when reaching the end of the application execution.

6.4.1 Data collection

Traffic coming from probes should be kept ideally distinguished from traffic coming from cores, in order to avoid influence on normal system communication. As reported in [189], three main options can be considered: *Separate Physical Interconnect for the original NoC application and the NoC Monitoring Service* (i.e., implementation of a *service network* [138, 137]), *Common Physical Interconnect but Separate Physical NoC Resources* (i.e., use of virtual channels), *Common Physical Interconnect and Shared Phys-*

ical NoC Resources (i.e., use of bandwidth regulators or QoS techniques for separating the traffic [9]). We have chosen in our study to consider an NoC in which communication resources are shared among the two types of traffic. This solution is particularly convenient in our case, being the overhead on the NoC implementation limited, but the network should be able to guarantee enough bandwidth to support both data flows without significant interference between them.

In general, the communication system should be over-designed in order to allocate the monitoring traffic together with the NoC traffic, and avoiding significant influences between the flows. As shown for instance in section 6.5, this can be done a posteriori by verifying that data forwarded to the PMU represents a negligible part of both the traffic generated by the NoC and the total bandwidth available, or by limiting the bandwidth available to probes.

To evaluate the influence of monitoring activities on the data collection, we performed an analysis of the traffic generated by the probes. A similar consideration holds also for the data storage. Table 6.4 presents the bandwidth expected to be generated by a selected set of events, in terms of the execution time of the application (*execution_time*), the length of the time window observed (*time_window*), the sampling time of the queue utilization (*queue_sampling_time*), the number of transactions executed during the application (*number_transactions*), and the aggregation factor (α). Similar formulas can also be found for other events not shown in the table.

The minimum traffic generated by the probe is equivalent to 64 bits, i.e., a 2-flit packet composed of a header and a payload flit (we assume a 32-bit data width for the NoC physical links). Such traffic can be observed for single event messages, such as those generated in the case of the detection of the measurement of the throughput during the execution of an application. A 96-bit packet (three flits) is employed for sending information about averaged measurements. The first data flit contains the measured value, while the second the counted number of occurrences of the event.

For some measurements, such as the one counting the number of times the initiator-to-initiator (I2I) latency is above a certain threshold, the bandwidth depends on the number of occurrences of the event. Table 6.4 reports therefore the maximum value, i.e., the bandwidth generated when all the transactions are above threshold.

The overall bandwidth of the probes can be calculated once the values of the monitoring parameters have been set (*execution_time*, *time_window*, *queue_sampling_time*, α). In the case of messages depending on the number of events appearing on the NoC (such as for instance in the case of the notification of the messages length, or in the case of the number of transactions generated during a dynamically changing application) an estimation of the needed bandwidth should be performed before data collection. This can be done for instance by preceding the measurement of the event to be estimated with another execution of the application, during which the probe measures the number of occurrences of the event.

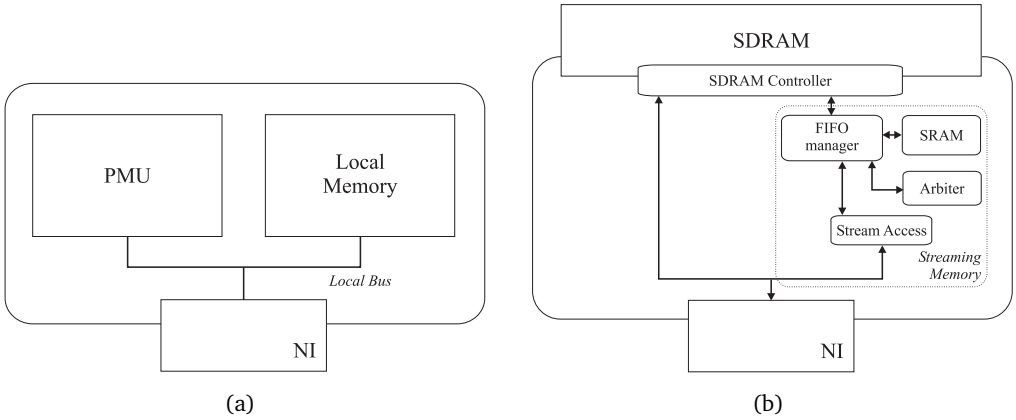


Figure 6.8. Memories used for storing the collected data: (a) PMU local memory; (b) streaming memory.

6.4.2 Storage

In terms of the storage strategy adopted in our architecture, we can distinguish between two approaches: the use of local memory in the PMU, and the use of an on-chip streaming memory for larger amount of data. Those two different approaches can support the run-time management the and profiling features, respectively. Figure 6.8 shows an overview of the two approaches.

PMU Local memory

The former approach, shown in figure 6.8(a), can be employed for events generating a limited number of values for each execution (such as for instance the throughput measured in one communication, the average message length, or the number of transactions). Local storage is moreover important in the case of utilization of collected data for the analysis of run-time system behavior for run-time management of resources, or in general for applications that may require an adaptive reaction from the system. As a matter of fact, values stored in the PMU's local memory or registers can be accessed by the Operating System in a faster way, and they can be employed for evaluating appropriate reactions to changes in the system detected by the probes.

Streaming memory

The streaming memory approach is employed for storing data exceeding the allocated space in the PMU's local memory, and for data whose dimension is not known before the execution of the application to be monitored. The streaming memory can be implemented by using an approach similar to [199], where a real-time streaming memory controller supporting off-chip services and real-time guarantees for accessing exter-

nal memory is proposed. The need for using a streaming memory in our monitoring framework lies on the fact that it allows to store streams of data without taking care of the addresses generation, and to use as much as possible the memory bandwidth. In our architecture (see figure 6.8(b)), a FIFO manager is in charge of retrieving data from the input buffer receiving the streaming data of the probes, and of implementing the address generation for the streams, as well as updating the access pointer to access the external memory. The buffer is memory mapped to the probes through the *Storage_address* field in their configuration registers. The usage of different memory addresses to map the same input buffer is enabled to manage the record of different streams. The PMU will read the data stored once the application is completed, using the inverse (*read*) functionality of the FIFO manager. The header of the message is stored in the memory along with the information generated by the probe, in order to let the PMU be able to associate the information with its source when performing the elaboration of collected data.

For systems with a large number of probes, only one streaming memory might represent a bottleneck for the monitoring system, both in terms of the possible traffic saturation and in the waiting time for accessing the memory. More streaming memories can be envisioned and therefore distributed in the NoC to solve this issue. Extending the PMU for using more streaming memories is straightforward, as far as they are memory-mapped to the centralized control management unit, and as far as additional interfaces to the main memory are available, as for instance in 3D-stacked memory architectures [200].

6.4.3 Probes Management Unit

Depending on the purpose of the monitoring system, the PMU can be in charge of managing the profiling or of the run-time adaptation of the resources. In the first case, tasks performed by the PMU are mainly two: programming the configuration registers, and retrieving the collected data for processing them. These tasks can be implemented with two software routines running on a processor. No particular performance requirements are needed concerning these two routines, as they are executed when the application is not running. The former task is executed before running the application to be monitored, and it is in charge of writing the memory-mapped configuration registers of each probe for detecting the desired event. A record is kept by the PMU that associates the register identifier with the configuration written in it and the memory address where to retrieve the stored data.

The second software routine is executed at the end of the application, and it reads the stored data so as to process them in order to perform additional operations on the collected information, such as the calculation of the throughput or the averaging of the measurement performed. These two routines can be executed by any processor before and after the execution of the application. With this choice, no overhead cost is associated with the implementation of the PMU.

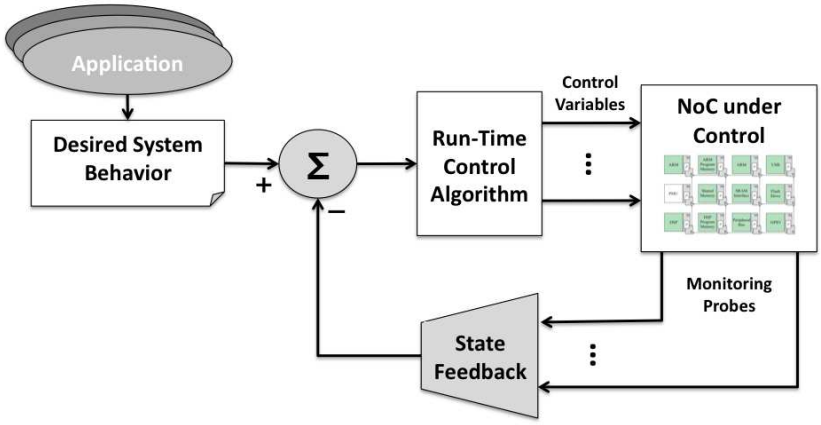


Figure 6.9. Run-time control loop based on the proposed NoC monitoring framework.

In the case of the use of the PMU for the run-time management of the platform, a third software routine is active on the core selected as run-time manager [134, 195]. The run-time manager routine is in charge of analyzing the data received by the several probes, elaborate them, and to react appropriately to changes in the application requests by appropriately configuring the utilization of system resources, such as the operative frequencies of NoC link, buffer allocations, priorities of communication flows, etc [50, 133, 201]. Typical examples of utilization for run-time management issues of the NoC monitoring framework we are proposing, can be summarized in figure 6.9 where the feedback controller is used to set the desired system behavior. In this case, the controller does not need to be activated too much frequently since the period should be large enough to wait for the implementation of the selected policies by the control algorithm. Moreover, by using the *notify_PMU* field of the configuration register it is possible to generate an interrupt at the reception of the monitoring messages which immediately notifies the occurrence of the event to the PMU. This is particularly useful in the case of events such as the exceeding of a defined threshold in one of the measurements. An example of the utilization of the monitoring system for run-time management of NoC resources is given in subsection 6.5.2.

General considerations about run-time management of platform resources

In the case of the use of the monitoring information collected by the probes for run-time purposes, the platform should provide hardware/software support for the execution of those tasks taking care of the reallocation of the resources, and of the modification of the operating points of the platform components. Mismatches between performance and quality of service required and those currently provided by the platform, due to parameters and workload run-time variations, should be minimized.

The decision about the adaptations to be performed on the platform can be taken at

different levels, i.e., at hardware, OS, or/and application, depending on the granularity of the adaptation and on the time overhead still allowing satisfying the requirements [202]. At system level, monitoring activity may target processors and memories behavior through dedicated hardware probes [203, 204, 202, 44] or by code structures in the OS and at application level [205, 206, 207, 208, 209], and, as also discussed in this dissertation, the communication subsystem behavior [140, 133]. Goals of the run-time adaptation range from reconfiguration of platform components [134, 18, 210, 204] to the reallocation of application tasks into the processing elements in order to, for instance, increase performance [211, 212], reduce energy consumption [213, 214, 215], or for fault tolerant purposes [216, 217, 218].

As presented in [45], several architectures have been proposed for the implementation of run-time management in MPSoCs. In general, it is possible to distinguish among three types of implementation, i.e., the *Master-Slave configuration*, the *Separate supervisor configuration*, and the *Symmetric configuration*. The first category includes all those systems in which the run-time management is executed by a single master processor, which monitors and assigns work to the slave processors. The master processor is in charge of collecting monitoring data and elaborating them, as well as driving the adaptation of the system. While being simple and efficient, this type of implementation may present the drawbacks of having in the master processor a single point of failure, as well as a possible performance bottleneck. Several works in the literature target this type of architecture: in [201], the run-time manager implements a state-space feedback control for tuning operating points of the voltage/frequency islands in which the architecture is divided; in [205], run-time management is employed for obtaining, in the case of changing application scenarios, a desired QoS while minimizing power consumption and maximizing the usage the multiple cores in the architecture; the Texas Instruments *OMAP* MPSoC platform [219] implements a software run-time manager running on a master processor and taking care of assigning tasks to the slave processing cores.

In the *Separate Supervisor configuration*, the run-time management functionalities are executed independently in each processor. A relatively costly synchronization among the processors needs to be implemented, while gaining in terms of scalability and graceful degradation in the case of the failure of one of the processors. This type of implementation presents however a significant overhead in terms of duplicated data structure, and inter-processor communication.

All the processors in the *Symmetric configuration* execute concurrently a single run-time manager. Shared data are accessed through critical sections. This configuration is implementable in the case of homogeneous shared-memory MPSoCs, and while being the most flexible architecture, similarly to the *Master-Slave configuration*, it presents downsides due to possible bottlenecks in the execution of the run-time manager. The *ARM MPCore* architecture [220] implements this type of configuration while running an SMP OS such as Linux SMP. Similarly, the *K42* OS [209] developed by IBM includes

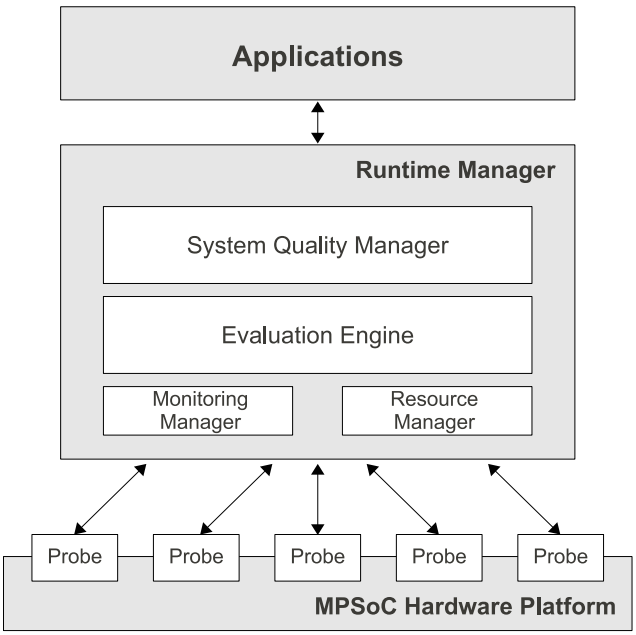


Figure 6.10. General structure of run-time management systems.

a software implementation of a Symmetric configuration run-time manager.

Mixed implementations exist which attempt to combine the above mentioned configurations, such as for instance the *StepNP* flexible multi-processor SoC architecture platform [139], which contains general purpose processing elements running an OS in Symmetric configuration that act as master for the slave dedicated cores and processors.

Figure 6.10 shows the general structure of a run-time management systems, in particular focusing on the case of the architecture considered in this chapter, i.e., the *Master-Slave configuration*. In the figure, the main functionalities offered and implemented in the case of run-time management are highlighted. Typically, applications may or may not be allowed to interact with the run-time management, by being able to configuring it or by taking over part of its tasks [45]. In our case, applications are considered separated from the run-time manager, which supports their adaptivity by selecting the most appropriate operating point according to resources requirements and availability. This type of architecture can be for instance observed in systems such as the one presented in [205] or [201].

The *Monitoring Manager* deals with the configuration of the probes and the collection of information. As presented in section 6.4.3, this is implemented in our system by running a software routine before the execution of the application, and by allowing the possibility to configure probes' memory-mapped registers at run-time. Collection of information is event-based, or performed at moments specified in the probes' configuration registers. The *Resource Manager* takes care of the configuration and assignment

of the resources to the applications requiring them, by trying to match availability with requirements. The *Evaluation Engine* calculates the new configuration to be applied to the system, by elaborating information detected by probes. Depending on the resources managed, different types of algorithms and heuristics can be used for optimizing the new configuration, in particular trading-off execution speed versus quality of the solution. As an example, state-space feedback control is employed in [201], while heuristics in [213, 214, 212, 221]. Evolutionary algorithms can be employed for calculating the optimal distribution of resources [222, 223, 224], but their execution can be to costly to be performed at run-time. Systems based on the storage of pre-calculated solutions have been proposed [217, 204], that provide the run-time manager a pool of possible configurations to choose from for satisfying new requirements. The *System Quality Manager* interacts with user, applications, and *Evaluation Engine* in order to find the right trade-off between the applications and user requirements, and the available platform resources [45].

6.5 Experiments

6.5.1 Profiling of ray tracing application

In order to evaluate the overhead of employing the monitoring system in a real case study, we implemented a cycle accurate simulator of an NoC in SystemC. In our experiments, we considered the architecture shown in figure 6.1, composed of 10 initiators, where each initiator has a processing element and 16KB local L1 cache, and a shared L2 memory. Cores of the architecture are mapped on a 4 x 3 mesh. As shown in figure 6.1, we located the L2 shared memory on tile (1,1), while tile (1,2) was assigned to the Probes Management Unit. As use-case, we monitored the execution of a *ray tracing* application [225]. *Ray tracers* are typically used to render scenes in games, 3-D modeling/visualization, virtual reality applications, etc., and they can be considered as one of the key challenging applications for general-purpose and embedded processors [226]. The application was parallelized and mapped on the ten processing elements. Every processor generates read or write operations toward the shared memory, driven by the memory request generated by the application. We deployed 4 probes in each initiator tile, and perform with each probe a different measurement within the same node. Configurations of the probes, a part from the *addr* fields, were set to the same value for each node. We measured the total traffic generated by each initiator towards the target, the average initiator-to-initiator (I2I) latency, the number of times the I2I latency exceed the threshold (set to 95 cycles), and the throughput generated during each time window (set for this experiment to 2^{15} cycles (32Kcycles)). We ran the simulation for 15 Mcycles, starting to gather our measurements after a warm-up time of 10 Kcycles.

Figure 6.11(a) and figure 6.11(b) show respectively the traffic measured from each

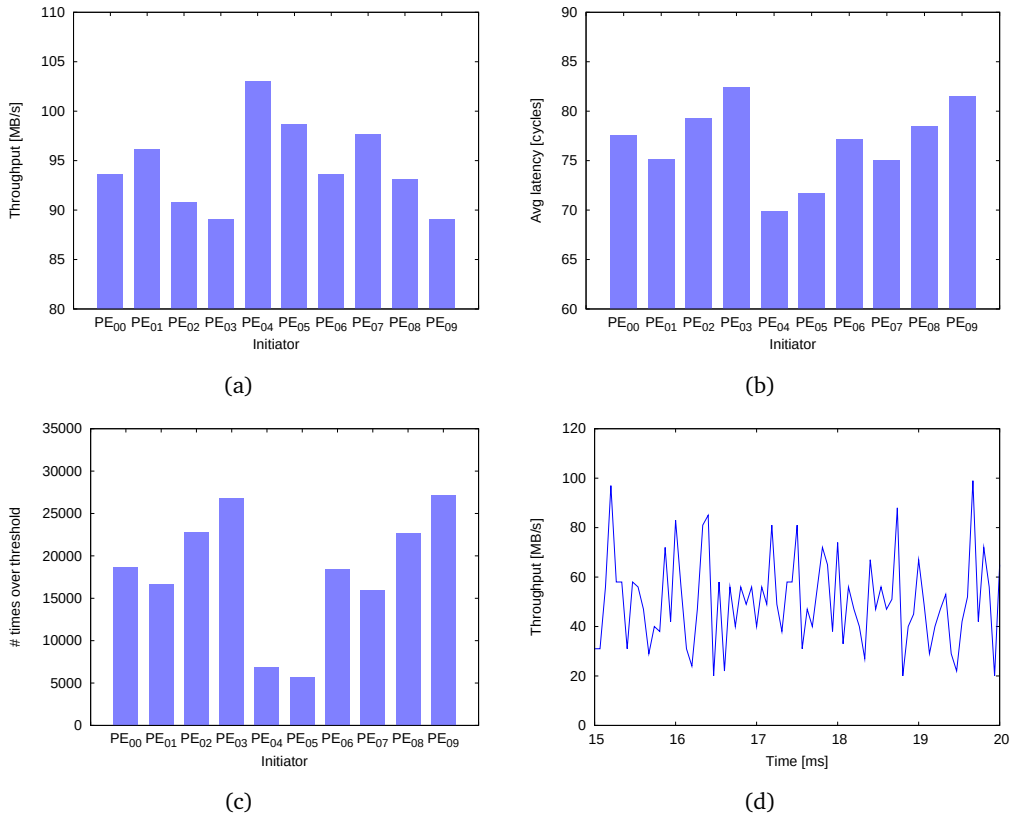


Figure 6.11. Monitoring results: (a) Traffic from each initiator to the shared memory (MB/s), (b) Average I2I latency from each initiator (cycles), (c) Number of times I2I latency over threshold, (d) Throughput detected for initiator in PE_1, in tile (0,0) (MB/s).

initiator to the shared target (in MB/s), and the average transaction latency (I2I) for each connection (in cycles). Figure 6.11(c) shows the number of times in which the I2I latency exceeded the value set as threshold. By using the threshold capability the number of messages sent by the probes is reduced in average of about 91%, reducing accordingly also the bandwidth needed for the transmission of the messages.

Figure 6.11(d) shows an extract of the throughput (in MB/s) generated by *initiator 1* (in tile (0,0)) while varying the time. The graph shows the values of the traffic detected at every time window, and it provides a good example of the possibility to observe with our monitoring system run-time characteristics of the application. The throughput generated by the probes during the experiment, averaged on the number of initiators, is shown for the measured events in table 6.5. The first two values are of fixed data length and the probe send only one message. The second two values depend on the run-time parameters of the execution. In particular, for the detection of the throughput, only one packet, composed of the header and one payload flit, is

Table 6.5. Throughput generated by each event in the experiments performed.

Event	**Bandwidth**
Throughput	0.26 KB/s
Average I2I latency	0.39 KB/s
I2I latency over threshold	4.62 MB/s
Throughput in time window	119 KB/s

Table 6.6. Average power consumption (in μW) associated with event detection and monitoring data transmission at the *initiator 1*.

Event	**Detection (μW)**	**Transmission (μW)**
Throughput	78.911	0.023
Average I2I latency	194.699	0.0344
I2I latency over threshold	210.316	419.668
Throughput in time window	79.303	10.432

generated. For the measurement of the average I2I latency, the packet generated contains two payload flits. In the case of the measurement of the number of transactions with I2I latency above the threshold, the throughput generated (and the number of messages sent by the probe) depends on the number of events detected, while for the throughput measured in every time interval, the amount of information transmitted is inversely proportional to the dimension of the time windows. The traffic generated by the probes is about 5% of the traffic generated by the initiators (measured with the *throughput event* detector) and 0.2% of the NoC link bandwidth (equivalent to 2GB/s). These values allowed us to verify in the experiments performed the assumption of non intrusiveness of the monitoring system.

Table 6.6 shows the values of the average power consumption (in μW) associated with the detection of the four events at *initiator 0* (PE_{00}), as well as the average power consumption due to the transmission of the monitoring data. The average power consumption related to the detection were calculated considering the values reported in table 6.2, and the rate in which the event detectors were active during the execution of the application. On the other hand, the average power consumption associated with the transmission of the monitoring data has been obtained by using the values shown before in table 6.3 for routers and NIs, the number and type of routers encountered by the monitoring messages from the probe to the PMU and related bandwidth. As table 6.6 shows, the power consumption related to the detection of latency values is higher than the one related to the detection of the throughput. This is mainly due to the exchange of information between initiator and target for implementing the protocol needed for measuring the latency (see section6.3.1). The power associated with the data transmission is in general (except for *I2I latency over threshold*) lower than the one

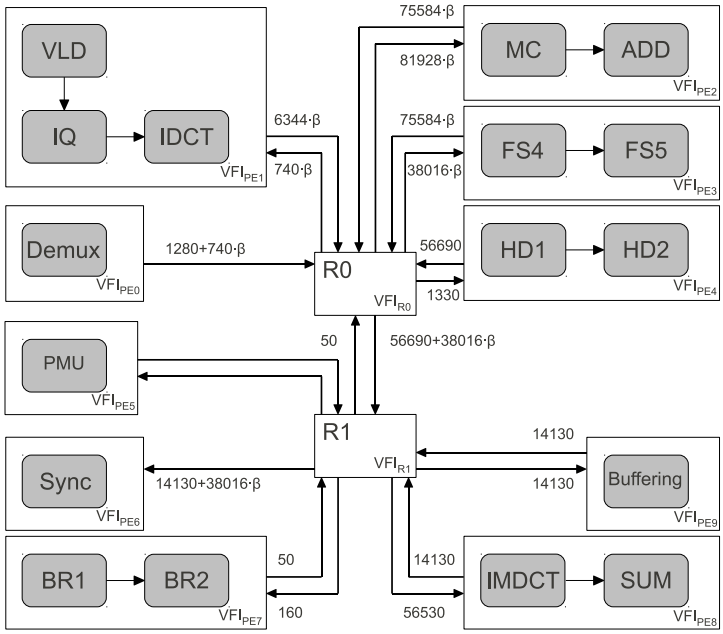


Figure 6.12. Architecture and communication bandwidth of the MMS. The value of β depends on the application mode. $beta = 0$: Only audio stereo (AS_V0); $\beta = 1$: Audio plus low resolution video (AS_VL); $\beta = 2$: audio plus medium resolution video (AS_VM); $\beta = 8$: audio plus high resolution video (AS_VH).

associated with the detection, due to the fact that for the measurements performed the number of messages sent to the PMU is significantly lower than the number of transactions monitored. For instance, only one packet is sent by the probe in the case of the *throughput* and of the *average I2I latency*. Similarly to the results obtained in table 6.5 for the bandwidth required by the probes, the *I2I latency over threshold* measurement presents the highest value of power consumption, due to the relatively high number of messages sent by the probe. Overall, the average power consumption due to the monitoring system is about 6% of the power consumption due to the traffic generated by *initiator 1* executing the ray tracing application, that results equal to 19 mW.

6.5.2 Monitoring of queues utilization for dynamic voltage-frequency management

This section presents a use case in which the monitoring system is employed for providing information for adapting at run-time the voltage and frequency level of the NoC routers in a generic multimedia system (MMS). We consider a multimedia application including an H263 video decoder and an MP3 audio decoder [227]. Figure 6.12 shows how the tasks of the application have been mapped on the NoC architecture, as well

as the communication bandwidth of the cores (in KByte/s). Cores bandwidth depend on the parameter β, which varies accordingly to the video resolution of the application. Every processing element (*PEi*) and router (*Ri*) belongs to a different voltage and frequency island (VFIs). Without loss of generality we focus only on the adaptation of the operating points of the routers, by imposing the frequency of the PEs to 1GHz. The goal of the adaptive system is to optimize the voltage and frequency of the routers by monitoring the queues utilization, directly related to the waiting times of the packets in the queues and therefore on their latency [228]. We consider routers able to operate at the following discrete frequency levels: F = {1 GHz, 0.5 GHz, 0.250 GHz, 0.125 GHz, 0.0625 GHz}. The corresponding discrete supply voltage levels are based on the model presented in [229]. Queues connecting processing elements and routers are 8-slot long.

In our experiments, we consider four different application modes impacting the network workload: only audio stereo (*AS_V0*, $\beta = 0$), audio plus low resolution video (*AS_VL*, $\beta = 1$), audio plus medium resolution video (*AS_VM*, $\beta = 4$), audio plus high resolution video (*AS_VH*, $\beta = 8$). When moving from one case to another, the variations in the average utilization of the queues are detected by the probes, which communicate the event to the *PMU*. The PMU increases or decreases the operating levels of the routers, depending on the event detected. Queues utilization is measured every 32 cycles, and, in the case that the average of the measured values over the *time window* exceeds a defined threshold, a warning message is sent at the end of a time window to the PMU. The used *time window* is approximately 100μs long, and represents the control intervals in which the application is divided. This amount of time is conservatively large enough for completing the operations of event detection and communication to the PMU, execution of the simple control algorithm, and updating the routers with the new operating points [201].

We monitor the input queue of router R0 connected to the $MC \rightarrow ADD$ processing element ($QR0_{MC-ADD}$), being in fact $MC \rightarrow ADD$ the processing element requiring a higher bandwidth. Moreover, we monitor the input queue of router R1 connected to router R0 ($QR1_{R0}$), which gives an indication about the capability of R1 of satisfying bandwidth requirements of messages flowing within *tile 1* and of those coming from and directed to *tile 0*. By profiling the queues behavior in the four cases we obtained a simple rule for adapting the frequency of the two routers. The R0(R1) frequency will be increased when the utilization rate of the $QR0_{MC-ADD}$($QR1_{R0}$) is *higher than 0.3*, while decreased when the utilization rate is *lower than 0.05*. When the two alerts are received in the same control interval, priority is given to the adaptation of R1's frequency. More complex control algorithms based on analytical models of the queues behavior can be implemented [201, 228]. However, this is out of the scope of the focus of the work presented in this dissertation.

Figure 6.13 presents the results of our simulations, in particularly focusing on the moments when switching from one case to another showing a time window of 1.5msec.

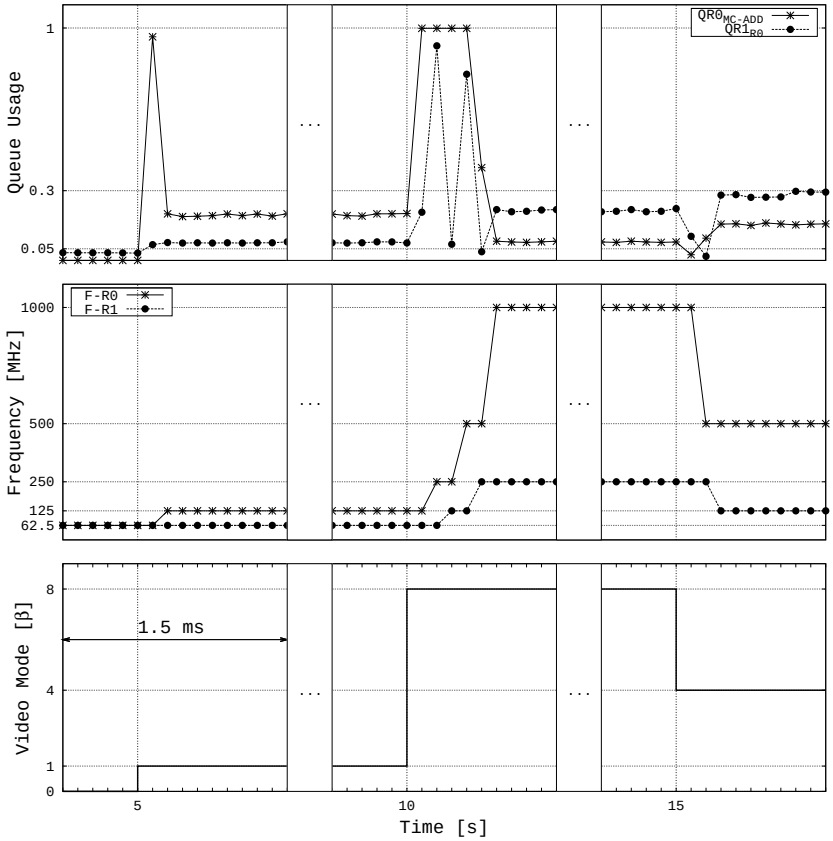


Figure 6.13. Monitored queues utilization and routers frequency during the application scenario composed of the following sequence: $AS\_V0 \rightarrow AS\_VL \rightarrow AS\_VH \rightarrow AS\_VM$. The results presentation has been focalized on the transition between application modes for a period of 1.5 ms.

We run every application mode for 5 seconds with the sequence $AS\_V0 \rightarrow AS\_VL \rightarrow AS\_VH \rightarrow AS\_VM$, for a total simulation length of 20 seconds. In figure 6.13 we show measured values for queue utilization and routers frequency when varying video mode (β). At second 5 and 10 the increment of the network load ($AS\_V0 \rightarrow AS\_VL$ and $AS\_VL \rightarrow AS\_VH$ transitions) can been noticed by the high queue utilization that requires a subsequent increment of the router(s) frequency, that in the second case occurs through 5 intermediate steps (0.5 ms). The opposite occurs at second 15 ($AS\_VH \rightarrow AS\_VM$ transition) where the under utilization of both queues required to reduce both frequencies. No other event occurs during the rest of the simulation. The average power consumption associated with the event detection and transmission is equal to 0.966 mW, while the traffic generated by the probes is equal to 72 Bytes. By employing the Orion [230] we evaluated the power consumption of our implementation with respect to a baseline architecture in which the routers have been designed

for the worst case scenario (AS_VH) running R0 at 1GHz and R1 at 250MHz, obtaining an average saving of about 19%, while the average saving increases up to 40% if comparing to the case in which both routers run at 1GHz.

6.6 Summary

In this chapter, we approached the problem of monitoring NoC based systems. Based on the study, presented in section 3, of the most common events detectable inside and through the communication subsystem, we proposed the utilization of a configurable multipurpose monitoring probe to detect information related to a large number of events.We discussed an efficient and automatic collection and storage of the information generated by the probes, and we evaluated the intrusiveness of the monitoring system, as well as the costs in terms of area, energy and traffic overhead of the proposed system. The resulting overhead can be considered reasonable given the provided service at network level both for profiling and run-time management purposes.

The work discussed in this chapter was presented in two conferences [195, 231].

Chapter 7

Fault tolerance

As previously discussed in chapter 3, new methodologies and architectural solutions should be explored in order to deal with malfunctions and failures due to the unreliability of complex devices and interconnects realized with new CMOS deep-submicron technologies. As shown in the motivation chapter, the network interface (NI) represents a critical point for the realization of a fault tolerant NoC. NIs interface IP cores to the overall system. Faulty NIs can cause errors that may affect the transmission of information within the NoC, as well as isolate the whole IP from the rest of the system, thus generating a massive and unwanted extension of the fault-affected area.

This chapter focuses on the proposal and evaluation of architectural methodologies to make NIs resistant to temporary and permanent faults. We will consider NoCs implementing a wormhole flow-control, and a source based routing. When a new packet is created, the NI's lookup table (LUT) provides, as routing information to be inserted in the header, a sequence of bits that encodes the path used by the packets to reach the destination node in the NoC. The length of this information segment depends on the dimension of the NoC and on its topology. At each router encountered along the path, a few bits of the sequence are employed for requesting the desired output port. After the use of the output port is granted, such bits are discarded, and the header of the packet is updated [9]. In the chapter we make the assumption that faults are local to the individual functional block of the NI; given the heterogeneity of such blocks, we accept distributions of multiple faults within an NI, imposing the single fault assumption within for the individual functional block. We consider mutually independent faults, and assume that the time between two consecutive faults hitting the same component is sufficiently long to apply at run-time the proposed online reconfiguration fault tolerant techniques described in this chapter.

The remainder of this chapter is organized as follows. Section 7.1 discusses contributions of this work with respect to the state of the art. Section 7.2 presents the proposed fault tolerant architectural solutions for the components of the NI, while section 7.3 discusses online error detection and run-time reconfiguration policies. Sec-

tions 7.4 and 7.5 describe the evaluation of the proposed NI architectures, showing that our solution is able to increase the fault tolerance of the NI to values close to those obtained for a reference standard triple modular redundancy (TMR) implementation. While adding a limited overhead with respect to a baseline NI architecture, our solutions achieve an area saving of up to 48% with respect to the TMR implementation, as well as a significant energy reduction.

7.1 Contributions with respect to the state of the art

The work presented in this chapter can be considered complementary to previous work about fault tolerance in NIs and NoC, as presented in chapter 4. With respect to it, we address not only the "hard" faults in the link connecting the core to the NI, but we proposed a solution able to deal with "hard" and "soft" faults in all the main architectural elements of the NIs. In this work, we address tiled architectures (such as for instance the one in [3]), in which the link between core and NI can be considered as part of the node's circuits and signals, and treated accordingly with standard fault tolerant techniques. For this type of NI architecture, a careful analysis of the fault tolerant capabilities of the NI has not been performed up to now; often, a "collapsing" of NI and core is adopted as far as faults are concerned. This work provides an evaluation of possible architectural techniques to be used for increasing fault tolerance characteristics of the NI's main components, and, therefore, of the overall NoC.

The main contributions of this chapter are:

- The proposal and evaluation of a fault tolerant two-level architectural approach for NI's components identified as most sensitive to faults, i.e., FIFOs or buffers, the lookup table (LUT), and the Finite State Machines (FSMs) driving NI's operations. The solutions proposed and discussed require a limited amount of redundancy and yet are able to mitigate the effects of both temporary and permanent faults in the NI.
- The proposal and discussion of online reconfiguration strategies for the fault tolerant components, activated as a consequence of fault detection.
- A methodology for the exploration of fault tolerant architectures by taking their *survivability* as one of the cost metrics.

7.2 Proposed NI fault tolerant approaches

Goal of this work is to improve NI's resilience by increasing fault tolerant capabilities of its basic blocks. As shown in chapter 3, in our fault model errors are mainly due to faults concerning the *Lookup Table* (LUT), *FIFOs*, and the *Finite State Machines* (FSM) controlling the adaptation protocol in the OCP adapter and the NI back-end operations

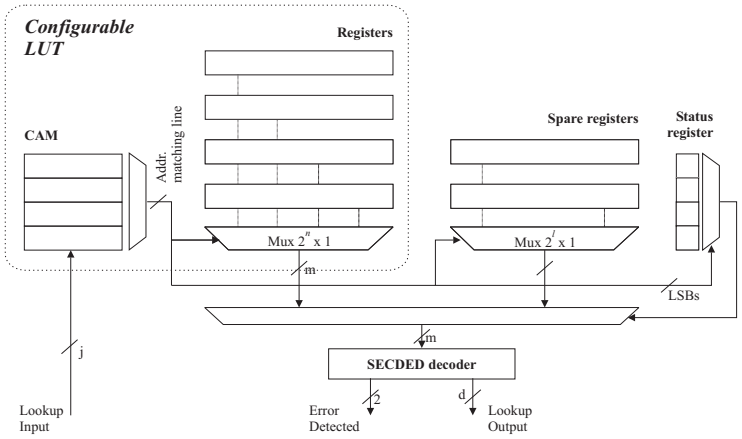


Figure 7.1. Overview of the proposed LUT architecture.

(data packetization/depacketization, routing and control flow). We therefore focus on these building blocks. These NI's components are mainly composed of memory cells (SRAM or Flip-Flops) and they are particularly subjected to both "soft" and "hard" faults [63, 65].

The solutions presented here for implementing these basic components are based on the use of Error Correcting and Detecting codes [61], in combination with limited redundancy, and a limited use of TMR. More specifically, additional logic and components of the NI, which represent a not significant part of the overall NI architecture area (in our implementations, around the 4% for a NI of a 16-node NoC), are implemented using TMR.

7.2.1 Lookup table

In a source-based implementation of the NI, a LUT is employed for retrieving the routing path associated with the address specified in a transaction [10]. Faults in the information stored into the LUT may generate mainly *Routing Path Errors*, which represent the majority of the errors appearing in the fault injection campaign presented in chapter 3.

As baseline architecture, we consider a LUT implemented as a combination of a non programmable Content-Addressable Memory (CAM) [182] and RAM or a set of registers (labeled *Configurable LUT* in figure 7.1). Without loss of generality, in this chapter we refer to a register-based implementation. The CAM contains hard-coded the address boundaries of the memory-mapped components of the NoC. When initiating a new transaction, the most significant bits (MSBs) of the operation address are compared with the values coded into the lines of the CAM. The position of the CAM line matching the input address is used to select the register in which is stored the output of the lookup operation, i.e., the routing path to reach the destination node mapped to

the input address. Routing path information are stored into the LUT's registers at boot time or after topology reconfigurations.

Figure 7.1 shows the architecture proposed for increasing the fault tolerance of the LUT. For the sake of clarity, in figure 7.1 we only show the architectural elements related to the lookup operation. We implemented a two-level approach which employs Error Correcting and Detecting Codes and a limited amount of architectural redundancy, allowing us to deal with both temporary and permanent faults in the LUT. Path information are stored by using a Single Error Correcting and Double Error Detecting (SECDED) Hsiao code [232] that is able to correct up to one error and detect up to two errors in each LUT's register. A Hsiao encoder encodes the information when writing the register, while a decoder decodes it after lookup.

The Error Correcting Code (ECC) corrects single-bit soft errors as well as hard ones. However, a hard error will recur every time the bad cell of the register is used, and, as hard errors accumulate, the device may become slowly unusable. In order to provide architectural redundancy to the LUT, we included in the design a certain number of spare registers that are meant to substitute LUT's registers in which the number of faults is higher than one, and that cannot therefore be anymore employed for storing correctly the routing information. These spare registers are of critical importance because a defective LUT register will cause an entire core not to be reachable from that NI. A bit in a *Status* register specifies whether a specific register of the LUT is working or faulty, by selecting either the nominal of the spare register. Spare registers are simply addressed through the least significant bits (LSBs) of the addressing signals. We implemented the *Status* register, as well as all the control logic, by using TMR. Spare registers can be employed for substituting faulty LUT's registers both during the post-manufacturing testing phase and at run-time for online faults due to the wear-out of the device.

The use of row/column substitution is a well known technique extensively used in industry for dealing with hard errors in RAM memories [233]. Similarly, ECCs are employed for error correction and detection in large memories. However, for relative small storage elements such as the LUT of the NI, the usage of these techniques must be carefully planned and adapted to a constrained environment, in order to avoid the large overhead associated with them. Moreover, lines substitution is mainly used for replacing elements found faulty during the off-line testing, while in our case, as explained in detail in section 7.3, we address substitution at run-time.

The presented LUT architecture allows protecting the NI from *Routing Path Errors* by detecting and correcting the degradation of the routing information, both in the case of temporary and permanent faults. *Routing Path Errors* are also avoided by protecting the control registers of the LUT through their TMR implementation, reducing the probability of selecting the wrong routing path register associated to an input destination address.

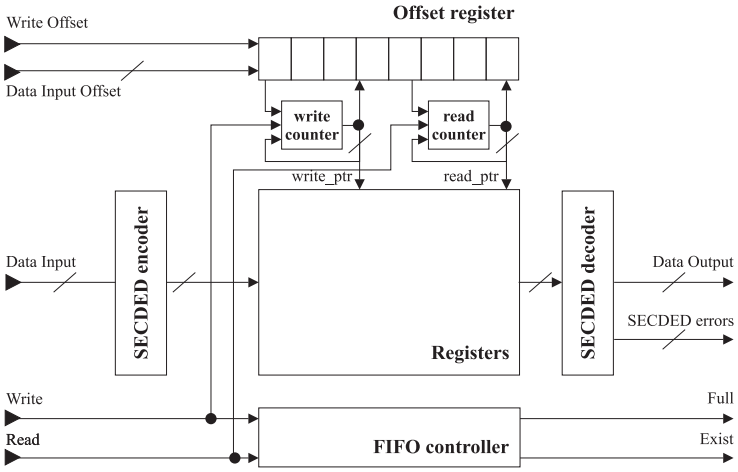


Figure 7.2. Overview of the proposed FIFO architecture.

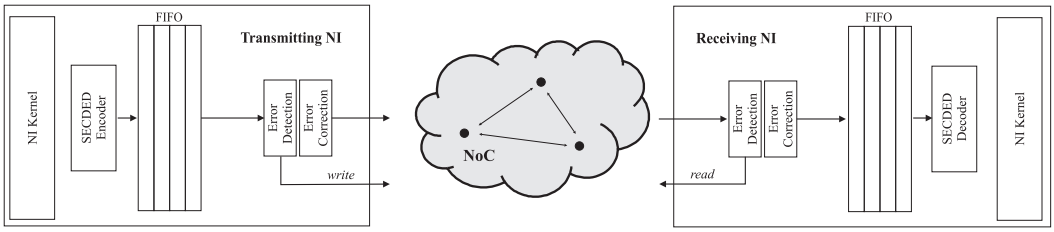


Figure 7.3. Integration of the proposed FIFO architecture within an NoC link implementing error correction and detection.

7.2.2 FIFOs

FIFO queues in NIs are used for decoupling computation from communication, in order to prevent the stall of the IP blocks due to the communication interconnect, and for allowing a separate implementation and optimization of the two system components [5]. Different FIFOs implementations have been proposed in the literature [198, 234]. Our baseline architecture is a register-based synchronous FIFO circular buffer, but the same considerations hold also for FIFOs implemented by using Dual-Port RAMs. We consider a FIFO whose data-path is equal to the flit's dimension (data and control signals). As figure 7.2 shows, in addition to the storage elements, logic is needed for managing the pointer to the element to be extracted (*read* pointer), the pointer to the first available position in the FIFO (*write* pointer), and for implementing control signals notifying whether the FIFO is full (*Full*) and whether at least one element is present in the FIFO (*Exists*). The *read* and *write* pointers are implemented as counters which are updated depending on the write or read operation performed on the FIFOs.

The implementation of fault tolerant FIFOs has been recently addressed by related work on NoCs. In [235], a reconfigurable buffer is proposed, which can borrow ele-

ments from FIFOs in the neighboring router's ports, at the cost of an increased wiring complexity. The solution addresses however only permanent faults in the FIFO's slots, without discussing methods for detecting them online. Authors in [236] propose a buffer architecture that can intelligently adapt its operations for using as much as possible the least leaky slots, in order to react to process variations in the component. The buffer keeps a list of slots, populated offline, which is pre-classified in order of "leakiness". While providing a solution to process variation and energy consumption, the buffer does not address however protection from either temporary or permanent faults.

Figure 7.2 shows the architecture of the FIFO presented in this chapter for increasing fault tolerance in the NI. Similarly to the solution discussed for the LUT, the architecture presented employs a two-level approach. Information in FIFOs is encoded, one flit at the time, by using a SECDED Hsiao code. In order to deal with permanent faults in the component, we propose a FIFO architecture exploiting the intrinsic redundancy of the stages in the FIFO. In this way, we are able to provide also for this component a graceful degradation of the performance during its operations. As shown in figure 7.2, we associate with each slot of the FIFO an *Offset* register, which stores the offset to be added to the calculation of the next value of the *read* and *write* pointers for obtaining the next working slot in the FIFO. In the case of a completely non faulty FIFO, values stored in the *Offset* register are all set to 0 (next working slot is the one immediately following). In the presence of faulty slots, the register stores the value to be added in the calculation of next pointers' values for skipping them. The number of bits needed for encoding values stored in the *Offset* register varies accordingly to the number of adjacent slots that can be faulty at the same time. The *Offset* register, as well as the logic for generating the *read* and *write* pointers and the control signals are implemented in TMR.

Figure 7.3 shows the integration of the proposed FIFO architecture in an NoC implementing error correction and detection in the links connecting NoC's components [149]. With respect to the reference architecture [149], information is encoded before being inserted into the FIFO, and checked for errors when extracted from it. If no errors are detected, the coded information is directly sent through the link, bypassing in this way the encoding step before the link transmission needed in the reference architecture. Similarly, in the case of a single error, the coded information is corrected and directly transmitted. At the receiving NI, flits are checked for errors, corrected, and copied encoded in the receiving FIFO. When extracted from the FIFO, the decoder decodes the flits and provides them to the NI and the core.

By implementing the proposed architecture, we protect the NI from the corruption of the information stored into the FIFO. Functional errors such as *Corrupt Data Errors*, *Routing Path Errors*, and *Control Flow Errors*, due to faults in the FIFO slots storing, respectively, body flits, the header flit, or control flow information, are therefore avoided or significantly reduced.

7.2.3 FSMs

Protocol adaptation and NI kernel operations make use of Finite State Machines for controlling the correct behavior of the NI. The current state of the FSM is stored in a state register. A bit flip in one of the flip-flops of the state register of the FSM may generate unexpected results, or, even worse, bring the system to an indefinite state or to a crash.

Different techniques, based on some level of either hardware or information redundancy, have been proposed for reducing the sensitivity to faults of FSMs [237]. Due to the limited number of states needed both for the OCP protocol implementation and the NI kernel operations [8], FSMs in our baseline implementation are relatively small Moore state machines.

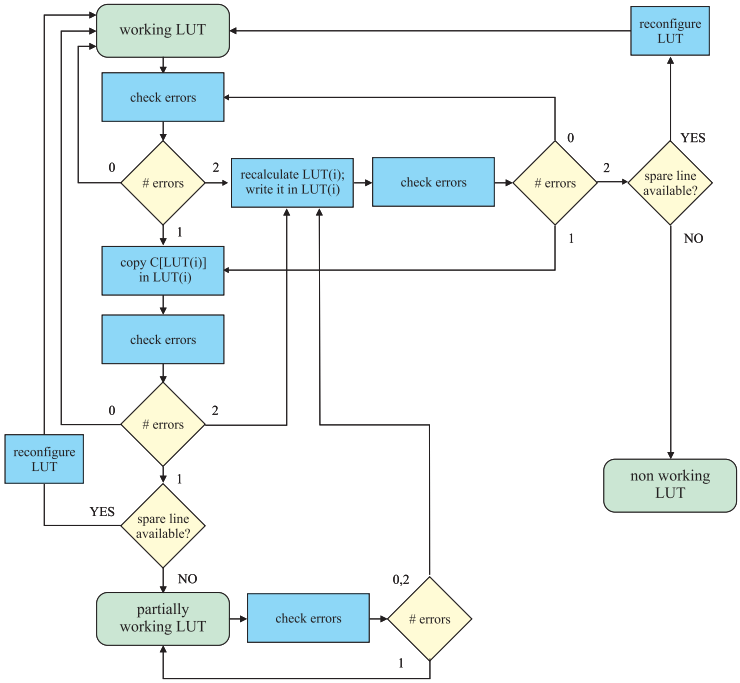
In our study, we adopt the SECDED Hsiao code for storing the state information of the FSM, and compare this technique with different implementations. In the discussed architecture, after calculating the next state of the FSM, the information is passed to a Hsiao encoder and stored in the state register. A decoder is used when retrieving the information about the state in the following clock cycle. Single errors in the state register are directly corrected by the decoder, while in the case of the detection of a double error, the FSM goes to a *reset* state, in order to avoid indefinite states in the system, while a warning signal is issued to communicate it to the NI and the core. A fault tolerant implementation of the FSM allows protecting the system from *Corrupt Protocol Conversion Errors*.

7.3 Error detection and reconfiguration policies

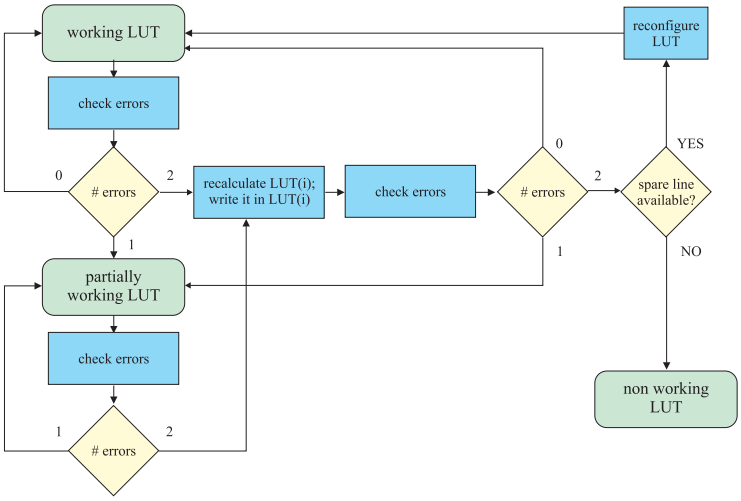
As previously presented, the solutions proposed are provided of a certain level of architectural redundancy for substituting elements (registers) permanently faulty. We employ the error-detecting characteristics of the Hsiao code for detecting permanent faults in the modules and for activating online reconfiguration by substituting at runtime the faulty elements with spare working ones. In this section, reconfiguration policies for the several modules are presented.

7.3.1 Lookup table

Figure 7.4 presents two possible reconfiguration policies that may be applied when detecting permanent faults in the LUT. We defined a *conservative* and a *non conservative* reconfiguration policy, which represent different trade-offs between the amount of offline time needed for reconfiguring the LUT, the time for recalculating the routing path stored in faulty registers, and the time for checking if the fault is permanent. In the reconfiguration flow presented in figure 7.4, a LUT is considered as *working* if, for each possible destination node, a register (either nominal or spare) where no faults have been detected exists. A LUT is *partially working* when one or more registers employed



(a) Conservative policy



(b) Non conservative policy

Figure 7.4. Diagrams describing the reconfiguration policies applied when detecting errors in the LUT.

for the lookup have one permanent fault: the LUT can still be used for providing a corrected routing information to the packet, thanks to the error correcting capabilities of the Hsiao decoder. A LUT is *non working* if it has more than one permanent fault in

at least one of the register, and no working spare registers are available for substituting it. In this case, the NI is not able to provide correct routing information for the packet directed to the NoC's node associated with the faulty register. The NI should be put offline, or the node memory-mapped to the address associated with faulty register should not be targeted anymore by the NI's communications.

As shown in figure 7.4(a), in the case of the *conservative* policy a system with *working* LUT reacts at the detection of a single error in one of the register by performing a check to determine if the error was caused by a permanent fault. The check consists in copying the information read from the LUT's *i* register, corrected by the SECDED decoder (*C[LUT(i)]*), into the same LUT register (*LUT(i)*). After copying the corrected data into the LUT, if the register still presents an error, the fault is considered as permanent. If no spare registers are available, the LUT is still used, but considered as *partially working*. In the case that working spare elements are still available, the LUT is reconfigured by enabling the use of the associated spare register, and still considered as *working*.

The detection of a double error requires to recalculate the routing path associated with the erroneous LUT register. The information recalculated is then copied in the register and checked again for errors. In the case of a confirmed double error, and of the not availability of spare registers, the LUT is considered as *not working*.

The *non conservative* policy shown in figure 7.4(b) reacts at the detection of a double error. It works as follows: in the case of detection of a single error, the correction capabilities of the SECDED are used for correcting single errors occurring during the following lookups; at detection of a double error, a check on the type of error is performed by recalculating the routing path stored into the LUT line, as done in the case of the *conservative* policy. As with the previous policy, in the case of a confirmed double error and if not spare lines are available, the system is considered as *not working*.

7.3.2 FIFOs

As described for the case of the LUT, policies can be defined for initiating the reconfiguration of FIFOs in the case of the detection of single or double errors in one of the FIFO's slot.

Figure 7.5 shows the *conservative* policy for the FIFO. A *partially working* FIFO has all the slots with at least one permanent error, but it still can be used because of the correcting capabilities of the SECDED code. The FIFO is *not working* when for at least one slot is not possible to skip all the adjacent faulty slots.

After detection of a single error by the FIFO's output decoder, the index of the faulty slot is recorded. When the same FIFO's slot is read again, it will contain a new flit. If the decoder finds again an error, the fault is considered to be permanent. The *offset register* is appropriately configured in order to skip in the following FIFO's operations the slot containing the permanent fault. In the case of detection of a double error, the slot is similarly checked for determining whether the error is a temporary or a

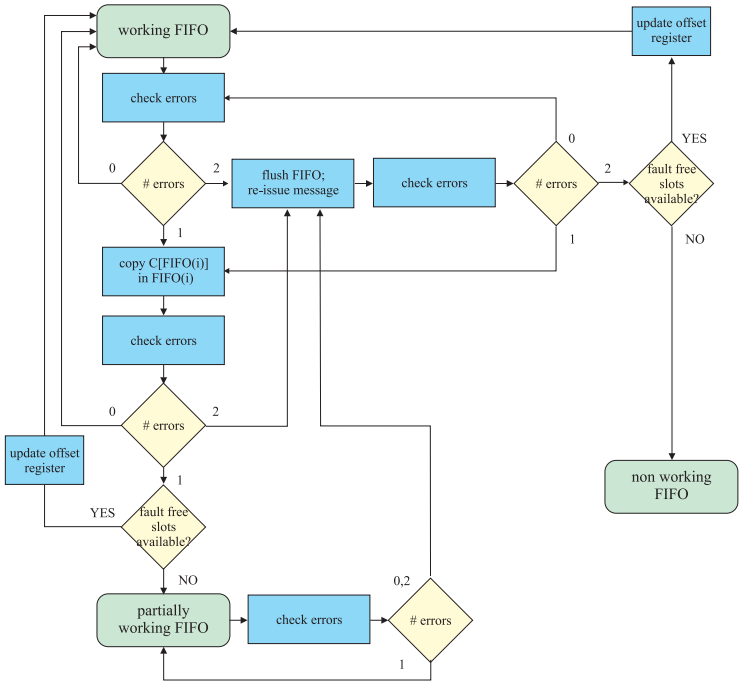


Figure 7.5. Diagram describing the conservative reconfiguration policies applied when detecting errors in the FIFO.

permanent one. In this case, the system should however flush the FIFO and reissue the packet, or rely on higher level data correction at the application level.

Similarly to the case of the LUT, a *non conservative* policy for the management of the reconfiguration of the FIFO tolerates the presence of a single error, while initiating the reconfiguration - therefore requesting a data retransmission or a reissuing of the packet - only at the detection of a double error.

Policies similar to those presented for the LUT and the FIFO could also apply to FSMs, for which the detection of a single error and double faults activates a check on the correctness of the information stored in the following FSM states on the status register, in order to determine if the fault is permanent or transitory.

7.3.3 Implementation of the policies

The reconfiguration policies are implemented as software routines running on the processor of the NoC's tile. When an error is detected, an interrupt request is generated by the NI. Additional signals specify the location of the fault (LUT, FIFOs, FSM, additional TMR components) and the type of error (single, double). When the interrupt is processed, the processor calls an interrupt handler, which reads the fault information and implements the reconfiguration policy for the component generating the interrupt request. The NI's interface to the core was extended to support the programming of

the configurable elements of the components, such as the *Offset* register of the FIFOs, and the *Status* register of the LUT.

7.4 Implementation results

In this section, we evaluate the architectural solutions presented in previous sections. In our evaluation, a *baseline* architecture is the reference architecture that does not implement any fault tolerant strategy. *TMR* architectures are implemented by employing TMR techniques. A *SECDED* architecture employs the Hsiao code for detecting and correcting errors, without implementing any architectural redundancy. We call *FT* those architectures employing the solutions previously described.

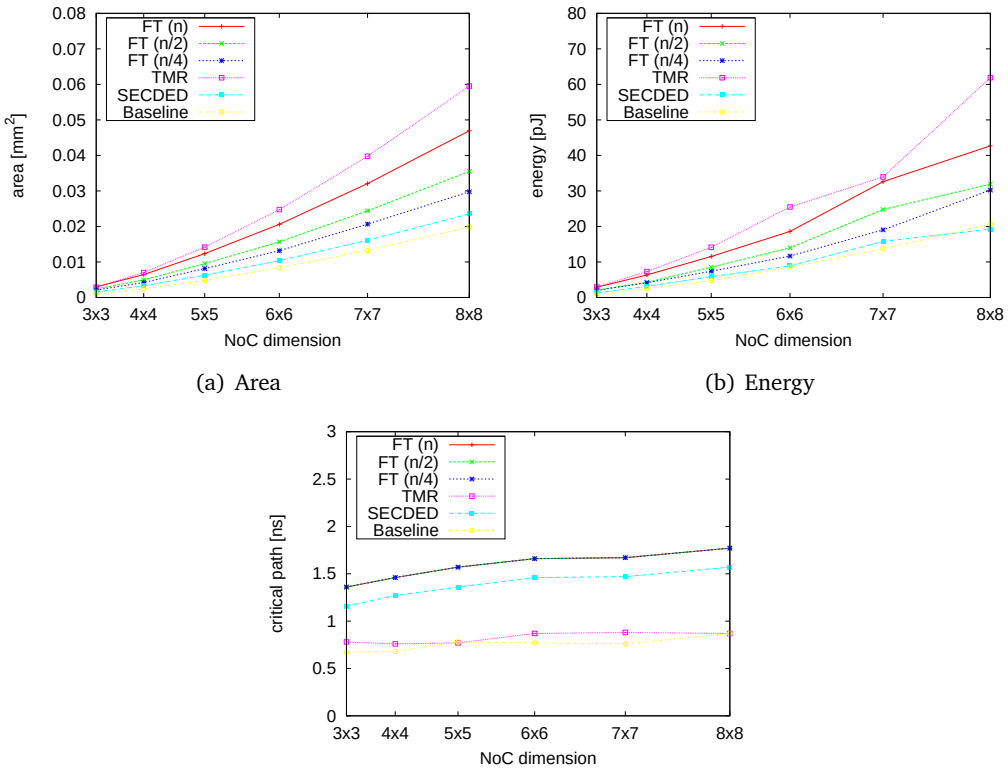
Without loss of generality, in our experiments we refer to an NoC with a square-mesh topology, and we employ routers with up to 5 output ports. We assume full connectivity between the cores, i.e., each core is able to communicate with all the other cores. We target an NoC architecture implementing a source-based routing. Each router encountered along the path to the destination node employs 3 bits of the routing path for selecting the desired output port [10]. For the considered NoC topology, we imposed the condition that the farthest reachable core is at $2\sqrt{n}-1$ hops, where n is the number of nodes of the NoC. The number of routing bits needed for encoding the path is therefore equal to $3(2\sqrt{n}-1)$. We consider a 34-bit data-path (32 bits for the data of the flit, and 2 bits for control signals).

We implemented components and NI in VHDL, and synthesized them by using Synopsys Design Compiler. We targeted the Nangate 45nm CSS typical open cell technology library [67]. Results shown were obtained by targeting the synthesis to a clock frequency of 500MHz. By using Synopsys Power Compiler, we obtained an estimation of the energy consumption associated with the operations performed by the components and the NI.

7.4.1 Lookup table

Figure 7.6 shows the area (in mm^2) of different implementations of the LUT, by varying the dimension of of the mesh, and, therefore, the number of registers needed for storing the routing information. The data length of the LUT's registers is equal to $3(2\sqrt{n}-1)$ for the *baseline* and the *TMR* implementation. In the implementations employing error correcting codes, namely *SECDED* and *FT*, the LUT's registers store along the routing path information also the parity bits. Given a number of bits d used for coding the path in each line of the RAM, the number of parity bits used for the SECDED is equal to $r+1$, where r is the minimum number satisfying the inequation $2^r \geq d+r+1$.

In the case of the *FT* architecture, we present results for implementations with number of spare lines equal to n, $\frac{n}{2}$, and $\frac{n}{4}$ (named *FT(n)*, *FT(n/2)*, and *FT(n/4)*,



(a) Area

(b) Energy

(c) Critical path

Figure 7.6. Area, energy consumption, and critical path of the different LUT architectures, while varying the dimension of the NoC.

respectively).

With respect to the *baseline* architecture, the *FT* architecture introduces an area overhead which varies with the dimension of the NoC. For the evaluated configurations, the maximum overhead was observed for the smaller NoC, i.e., the 3x3 mesh. Values of maximum overhead are 200%, 139%, and 110%, for a number of redundant registers respectively equal to n, $\frac{n}{2}$, and $\frac{n}{4}$. The area overhead decreases with the dimension of the NoC (136%, 78%, and 50% for the three FT implementations, in the case of a 8x8 mesh). The reduction of the overhead is due to the fact that the number of parity bits used for implementing the SECDED code is constant (equal to 7) in the configurations analyzed after the one with 25 nodes, and that the circuits for implementing the decoder and the encoder are shared by all the registers of the LUT. This overhead can be considered however acceptable, in particular if compared to the area measured for the *TMR* architecture. In the case of a the 8x8 mesh, our solution achieves a saving of up 50%, when considering for instance the *FT(n/4)* implementation.

As shown in figure 7.6(b), the energy (in pJ) shows trends similar to those obtained

when evaluating the area. When considering the *FT* implementation of the LUT, the maximum overhead (237%) with respect to the baseline implementation is obtained for the 3x3 topology, while obtaining a maximum saving of around the 51% with respect to the *TMR* (8x8 topology). Similarly to the values measured for the area, the maximum overhead can be observed for smaller topologies, while the saving with respect to the TMR implementation increases with the dimension of the NoC.

Figure 7.6(c) shows the length of the critical path for the different LUT's implementations, by varying the number of nodes in the NoC. As the figure shows, while reducing the amount of area and energy consumption, the *FT* architectures increase the critical path of the component. This is manly due to the fact that both in the SECDED and in the FT implementations a *decoder*, which in general is synthesized as a tree of XOR [69], is added to the critical paths of the LUT for decoding the information stored using the code. For high-speed circuits, a *TMR* implementation is therefore preferable, at a higher cost in terms of area and energy consumption.

7.4.2 FIFOs

In the case of the FIFOs, we analyzed area and energy overhead while varying the number of slots. In the *SECDED* and the *FT* architectures, information stored into FIFOs is encoded with a $(41, 34)$ SECDED Hsiao code. In the case of the *FT* architecture, we present results for implementations able to skip $m-1$, $\frac{m}{2}$, and $\frac{m}{4}$ faulty slots (named *FT(m-1)*, *FT(m/2)*), and *FT(m/4)*, respectively). m is the number of the total slots of the FIFO.

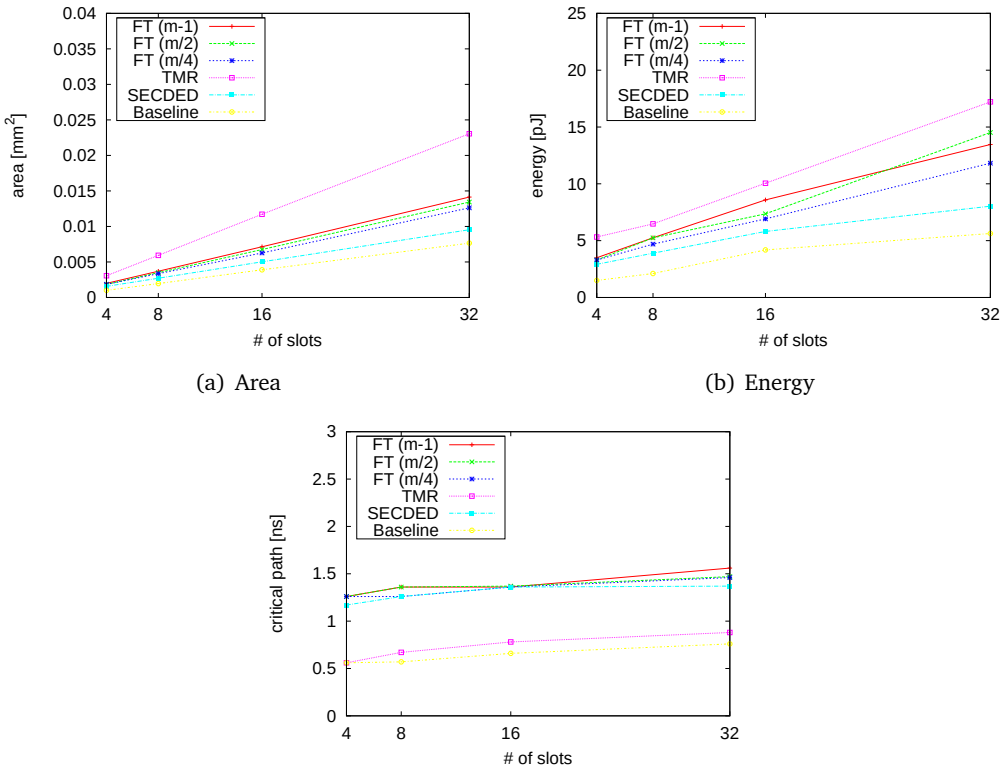
Figure 7.7 shows results obtained for the FIFO. The maximum overhead was obtained for smaller configurations, i.e., for the case of FIFOs with 4 slots. Values of maximum overhead are 98%, 83%, and 83%, respectively for the *FT(m-1)*, *FT(m/2)*, and *FT(m/4)*) implementations. The area overhead decreases with the number of slots (85%, 76%, and 65% for the three FT implementations, in the case of a 32-slot FIFO). With respect to the TMR implementation, the maximum saving in area (46%) is obtained in the case of a 32-slot *FT(m/2)* implementation.

In the case of the FIFO, the maximum energy overhead with respect to the *baseline* implementation is 149% (8-slot *FT(m-1)*), while the maximum energy saving with respect to the TMR implementation is 38% (4-slots *FT(m/4)*).

Figure 7.6(c) shows the length of the critical path for the different FIFO implementations. As already observed for the case of the LUT, the use of *SECDED* and *FT* architectures increases the critical path of the component.

7.4.3 FSMs

Information in state registers of the FSMs in the *SECDED* implementation is encoded using a $(7, 3)$ SECDED Hsiao code.



(a) Area

(b) Energy

(c) Critical path

Figure 7.7. Area, energy consumption, and critical path of the different FIFO architectures, while varying the number of slots.

Table 7.1 shows synthesis results obtained for the implementation of the FSMs. As it is possible to notice, values reported for the *SECDED* and the *TMR* implementations are comparable. As also discussed in [69], this is due to the fact that for registers of small dimension the encoder and the decoder used for implementing error correction and detection generate an area and energy overhead which is of the same order of the one obtained with a TMR implementation.

7.4.4 Network interface

Figure 7.8 shows synthesis results for the overall NI. The *baseline* NI was implemented by employing baseline version of FSM, FIFOs, and LUT. The *SECDED* NI employs SECDED versions of the FSM, FIFOs, and LUT, while a TMR implementation of the remaining components of the NI. The *FT(min)* configuration employs the *FT(n/4)* LUT and *FT(m/4)* FIFOs, while the *max* configuration the *FT(n)* LUT and *FT(m-1)* FIFOs. All the architectures employ 8-slot FIFOs. For the FSM, we considered the *SECDED* im-

Table 7.1. Area, energy, and critical path obtained by synthesizing the four implementations of the FSMs.

	Baseline	SECDED	TMR
Area $[\mu m^2]$	112.3	160.7	166.0
Energy $[pJ]$	0.060	0.110	0.149
Critical path $[ns]$	0.46	0.76	0.55

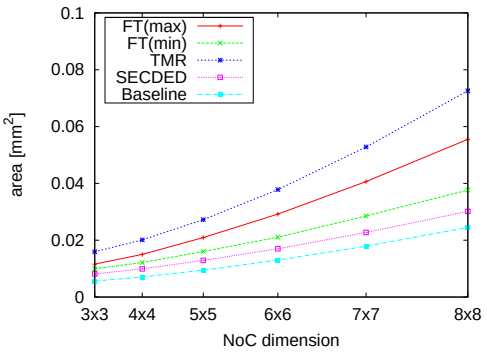


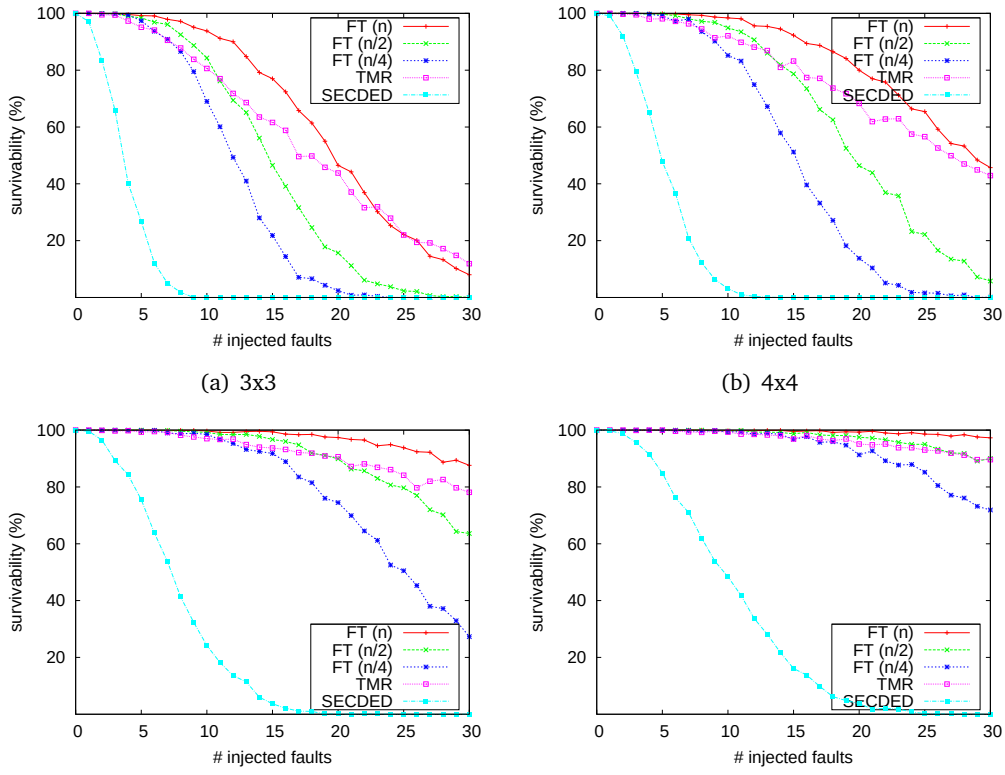
Figure 7.8. Area of different NI architectures, while varying the dimension of the NoC.

plementation, while the remaining NI's components were implemented by using TMR.

In general, the maximum area overhead can be observed for NoCs with smaller dimension, and it decreases when increasing the number of nodes in the topology. For a 3x3 NoC, the overhead was measured to be around 77% for *FT(min)* and 105% for *FT(max)*. The maximum saving with respect to a TMR implementation was measured for the 8x8 NoC (48% for *FT(min)* and 23% for *FT(max)*).

7.5 Survivability

In order to evaluate the effectiveness of the proposed solutions, we measured their *survivability*, defined as the probability of producing correct behavior in the presence of faults. We create a high level model of the NI and of its components in C++. The model simulates the behavior of each component at the occurrence of a new fault, by evaluating whether it is able to survive the fault or it produces an error. We evaluated the behavior of the system when a certain number of consecutive faults affects storage cells and circuit components. Each component of the NI is modeled by considering synthesis results about area and number of flip-flops in the design. By employing these models, we inject a defined number of faults in the component. Each single fault is injected at random time and in random position over the area. Injected faults are mutually independent, and enough time is left to the system to recover from the effects



(a) 3x3 (b) 4x4

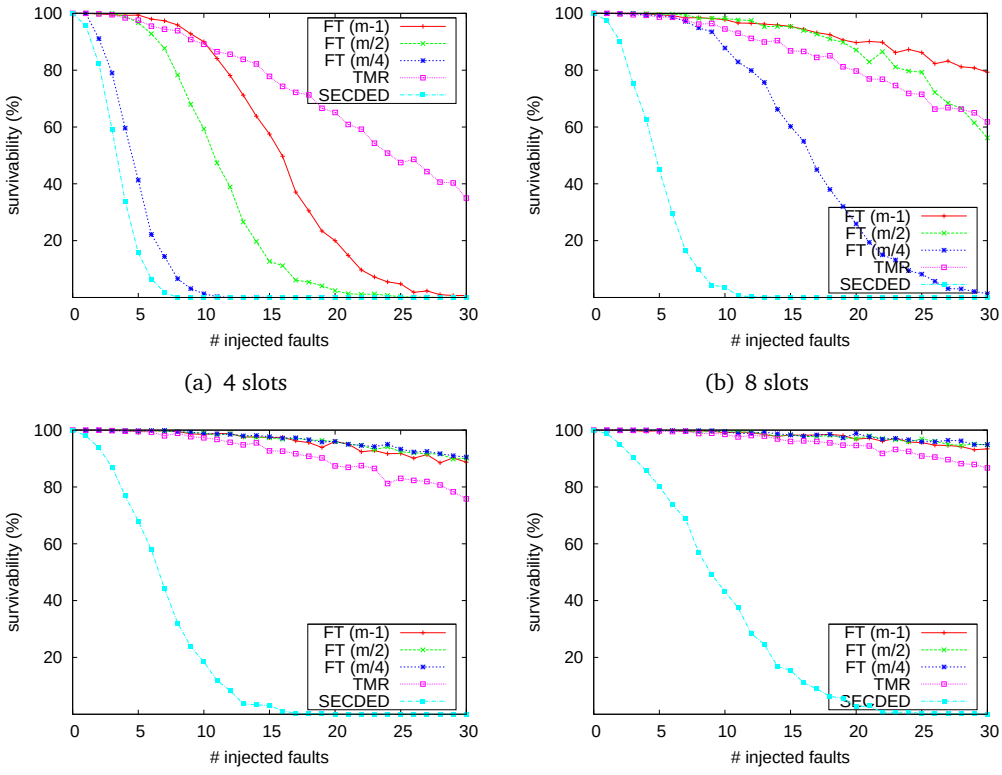
(c) 6x6 (d) 8x8

Figure 7.9. Survivability of the LUT for different NoC dimensions, while varying the number of injected faults.

of the fault, if any. For each fault, we simulate the behavior of the component and determine if with the injected sequence of faults it can be still considered error-free. For each amount of injected faults, we repeat the experiment 1000 times. Therefore, we count the number of times over the total experiments in which after the defined number of injected faults the errors generated in the component was non correctable or non detectable.

7.5.1 Lookup table

Figure 7.9 shows the survivability for the LUT, by varying the number of injected faults, in the case of different NoC dimensions. As the graphs shows, our solutions provide a survivability comparable to the *TMR*, while showing a significant improvement with respect to a *SECDED* implementation. For a relatively low number of injected faults, *FT* implementations are also able to provide better results that the *TMR* implementation. When the number of errors is too high, the *TMR* implementation of the NI provides however a better survivability. These results can be obtained thanks to the fact that



(a) 4 slots (b) 8 slots

(c) 16 slots (d) 32 slots

Figure 7.10. Survivability of the FIFO for different dimensions, while varying the number of injected faults.

FT implementations are able to correct up to two faults as long as enough redundant resources are available, while *TMR* implementations will only correct one fault. For higher numbers of errors, the number of spare resources available will not be sufficient for masking the fault. As figure 7.9 shows, the behavior of the survivability of the *FT* implementations with respect to the *TMR* implementation varies with the dimension of the NoC (i.e., of the LUT) and with the amount of redundancy. This fact can be explained by considering that for LUTs with bigger dimensions the probability of having a configuration leading to error-affected results for a given amount of injected faults is lower than in the case of smaller implementations. The survivability of the *FT* solutions outperforms the one obtained for the SECDED implementation. As shown in figure 7.6(a) and 7.6(b), this is achieved at the cost of an additional overhead in area and energy consumption (in average both of around the 96%, 53% and 29%, for the *FT(n)*, *FT(n/2)*, and *FT(n/4)* implementation, respectively).

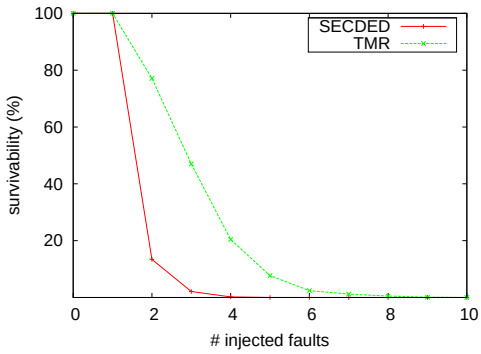


Figure 7.11. Survivability of the FSM implementations, while varying the number of injected faults.

7.5.2 FIFOs

Results of the analysis of the survivability for the FIFO architectures are shown in figure 7.10. In the figure, we show simulation results for different FIFO architectures by varying the number of slots and the number of faults injected. As obtained for the case of the LUT, for a relatively small number of errors the *FT* architectures outperform the *TMR* implementation. For reasons similar to those presented for the LUT, when increasing the number of slots in the FIFO and, therefore, its dimension, this result can be observed for a higher number of injected faults. The survivability of the *FT* solutions outperforms the one obtained for the SECDED implementation. The overhead in area and energy consumption of the *FT* implementations with respect to the SECDED implementation increases with the number of slots, and it was measured in a 32-slot configuration to be around 48% for the *FT(m-1)*, 40% for the *FT(m/2)*, and 32% for the *FT(m/4)* implementation.

7.5.3 FSMs

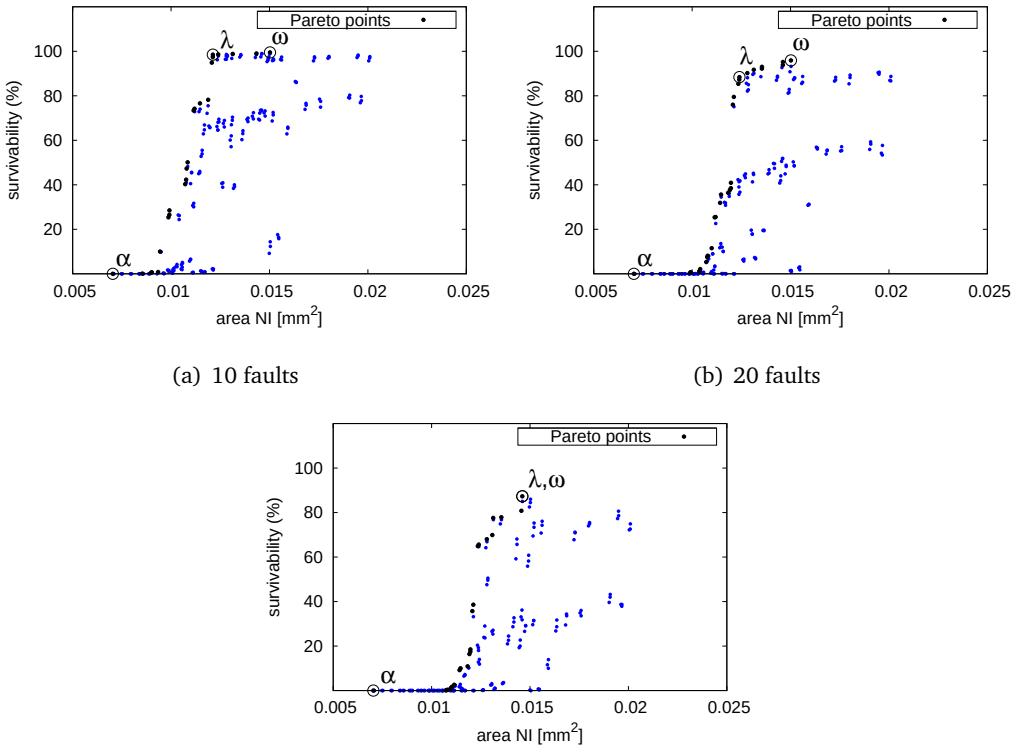
Figure 7.11 shows the results of the survivability analysis for the FSM. A TMR implementation is more resistant to faults than the SECDED implementation, confirming the fact that for the small registers of the FSM the use of error correcting and detecting codes is not convenient.

7.5.4 Network interface

In order to evaluate the survivability of the overall NI, we explore its resistance to faults when varying the implementation of its components. We performed an exhaustive multi-objective design space exploration of the several alternative NI architectures obtainable by combining the different implementations of the NI's components, as enumerated in table 7.2. We evaluated area and survivability of the configurations for a

Table 7.2. Design space for the NI.

Component	Architecture
LUT	BAS, TMR, SECDED, FT(n), FT(n/2), FT(n/4)
FIFO	BAS, TMR, SECDED, FT(m-1), FT(m/2), FT(m/4)
FSM	BAS, TMR, SECDED
other	BAS, TMR



(a) 10 faults

(b) 20 faults

(c) 30 faults

Figure 7.12. NI architecture exploration for different numbers of faults injected.

fixed number of injected faults. In the table, *BAS* refers to the baseline implementation, while the meaning of the other terms are the those previously presented in this section. In the exploration, we fixed to 16 the dimension of the NoC (a 4x4 mesh), and to 8 the number of slots of the two FIFOs of the NI.

Figures 7.12 show the results of the exploration when imposing a number of injected faults equal to 10, 20, and 30, respectively. Figures show also the Pareto points of the exploration. The Pareto configurations are those that in the design space exploration minimize the area and maximize the survivability of the NI. In the figures, we called α the point of the Pareto set with minimum area. Obviously, it corresponds to

Table 7.3. Characteristics and configurations of the α, ω, and λ Pareto points for different amounts of injected faults.

(a) 10 injected faults

	Architecture				Metrics	
Point	***LUT***	***FIFO***	***FSM***	***other***	***area (mm^2)***	***surv. (%)***
α	BAS	BAS	BAS	BAS	0.0070	0.0
ω	FT(n)	FT(m-1)	TMR	TMR	0.0150	99.5
λ	FT(n/4)	FT(m/4)	TMR	TMR	0.0120	98.5

(b) 20 injected faults

	Architecture				Metrics	
Point	***LUT***	***FIFO***	***FSM***	***other***	***area (mm^2)***	***surv. (%)***
α	BAS	BAS	BAS	BAS	0.0070	0.0
ω	FT(n)	FT(m-1)	SECDED	TMR	0.0150	95.9
λ	FT(n/4)	FT(m/2)	TMR	TMR	0.0124	88.4

(c) 30 injected faults

	Architecture				Metrics	
Point	***LUT***	***FIFO***	***FSM***	***other***	***area (mm^2)***	***surv. (%)***
α	BAS	BAS	BAS	BAS	0.0070	0.0
ω	FT(n)	FT(m/2)	SECDED	TMR	0.0146	87.3
λ	FT(n)	FT(m/2)	SECDED	TMR	0.0146	87.3

the NI architecture composed of the baseline implementation of all the components. As figures shows, its survivability is however 0%. ω is the Pareto point presenting the highest survivability. For 10 injected faults, ω is given by the NI architecture obtained when employing the *FT(n)* LUT, the *FT(m-1)* FIFOs, and by implementing the remaining components in TMR. For 20 injected faults, ω is given by NI architectural configuration similar to the one obtained for 10 injected faults, which employs a SECDED implementation for the FSMs. For 30 ω is given by an NI architecture composed of the *FT(n)* LUT, the *FT(m/2)* FIFOs, SECDED FSMs, and other components implemented in TMR.

Among the set of the obtained Pareto points, we selected those maximizing the product of *survivability* and $\frac{1}{area}$ (λ in the figures). The configuration corresponding to λ varies with the amount of injected faults. For 10 injected faults, the NI composed of an *FT(n/4)* LUT, *FT(m/4)* FIFOs, and a TMR implementation of the remaining components maximizes the chosen criteria. For 20 injected faults, the maximum value is obtained when employing an *FT(n/4)* LUT, *FT(m/2)* FIFOs, together with a *TMR* implementation of the remaining components. For 30 injected faults, as 7.12(c) shows, the ω point is the one the maximize the selection criteria. Table 7.3 summarizes values measured for area and survivability, as well the architectural configurations, of the α, ω, and λ Pareto points.

In should be noticed that for the fault configurations presented, none of the NI architectures included in the Pareto sets employs TMR implementations for the LUT

and the FIFOs. On the other hand, our *FT* architectures represents the best choice for the implementation of LUTs and FIFOs. These results where expected when observing in figure 7.9 and 7.10 the trend of the survivability of the single NI's components.

7.6 Summary

This chapter presented a study on the implementation of fault tolerant network interfaces. Chapter 3.3, by performing a fault injection campaign on the NoC components, demonstrated how the NI could be the main source of errors in the NoC, in particular when the number of nodes in the network increases. Soft and hard errors in the NI could cause an unwanted behavior that may create unrecoverable situations in the NoC, such as deadlock or livelock conditions. Based on the NI functional fault model proposed and discussed in chapter chapter 3.3, in this chapter we proposed new architectural solutions based on the use of error correcting and detecting codes and a limited amount of redundancy, and discussed policies for the reconfiguration of the components, which should be applied at the detection of errors. In our experiments, we obtained a saving of up to 48% in the area overhead, as well as a significant energy reduction, with respect to an alternative standard hardware TMR implementation of the NI, while maintaining a similar level of robustness to faults.

The work described in this chapter was published in [238, 239, 240].

Chapter 8

Conclusions and future work

In this dissertation, we demonstrate the possibility to provide system-level services to an NoC-based platform by enhancing the architecture of network interfaces with dedicated hardware/software modules. These modules, by working in parallel with protocol translation and data transmission, are able to support the high-level services at a relatively low cost in area and energy consumption. In this work, we focused in particular of three services: *security*, *NoC monitoring*, and *fault tolerance*.

In the first two chapters after the introduction, we introduced the reference NoC-based MPSoC architecture considered in this work, and we presented the motivations for focusing on the three mentioned services, based on experimental evaluations and quantitative results. In section 3.1, we presented an overview of security threats that may affect NoCs, as well as more general embedded systems, in particular showing how, without a mindful hardware design, simple software fallacies can give an attacker the possibility to access critical information, as well as disrupting significantly the services provided by the platform. Section 3.2 discusses motivations for implementing a monitoring system in the NoC that could help significantly both in understanding at design time the platform behavior, and in optimizing at run-time resource utilization. In section 3.3, we presented results of a fault injection campaign that we performed on the elements of the NoC (i.e. routers and network interfaces), demonstrating that network interfaces are critical elements from the point of view of the fault tolerance of the overall NoC, and that appropriate methodologies should be applied for their design.

In the three following chapters, we described the implementation steps and design choices for providing the system with the three above mentioned services, by adding custom modules to a baseline reference network interface architecture presented in chapter 2.

In particular, in chapter 5 we proposed a data protection mechanism for preventing illegal accesses to protected memory blocks in NoC-based architecture. The proposed mechanism is based on the use of dedicated hardware modules (called *Data Protec-*

tion Units) embedded within the network interface, that guarantee secure accesses to memory and/or memory-mapped peripherals by enforcing access control rules specifying the way in which an IP initiating a transaction to a shared memory in the NoC can access a memory block. We evaluated the proposed mechanism in terms of the area and energy consumption overhead in the case of two different architectures, showing a relatively low hardware implementation cost which is directly dependent on the number of IPs in the system and on the number of memory blocks to be protected. We therefore enhanced the protection system by proposing a methodology for the run-time management of the DPUs. Moreover, in the same chapter, we proposed a monitoring system, based on the DPUs, that can be employed for detecting attempts of illegal access to protected memory blocks as well as Denial-of-Service attacks.

In chapter 6, we re-targeted the monitoring system in order to deal with the detection of events happening within the NoC, and for providing support information for understanding system behavior and optimizing resource utilization. We detailed the implementation of a programmable multipurpose probe, that, embedded within the NI, can provide to a central management unit information about throughput and latency of NoC transactions, as well as utilization of buffers in NI and routers. We evaluated the intrusiveness of the monitoring system, as well as the cost in terms of area, energy and traffic overhead. The resulting overhead can be considered reasonable given the provided network level service both for profiling and for run-time management purposes.

The work described in chapter 7 deals with the design of fault tolerant network interfaces. New architectural solutions for the implementation of the main components on the NI were proposed, by applying a combination of error detecting and correcting codes and a limited amount of redundancy. Reconfiguration policies were proposed for exploiting architectural redundancy at run-time. Experimental results showed a significant reduction of area and energy consumption with respect to an alternative standard hardware Triple Modular Redundancy implementation of the NI, while maintaining a similar level of robustness to faults.

In this dissertation, we mainly focused on the evaluation of the support for the described single services. However, synergies can be found between the building components of the hardware/software support for minimizing the overhead in those designs in which more than one service is provided. All the services have in fact in common the propriety of monitoring a specific characteristic of the NoC, and to distribute the collected information to the global system in order to adapt the configuration or the behavior of the platform accordingly to its new run-time conditions. A part from the possibility of applying some of the studied fault tolerant techniques to the single building blocks composing the *security* and *monitoring* services, a common subsystem for collecting and distributing the information detected by the security and monitoring probes, as well as the presence of faults in the NI components, is likely to be shared amongst the services. Similarly, adaptation and configuration strategies can be de-

cided on the same *service management core* (either called *NSM* or *PMU* previously in the dissertation).

Future activities that it is possible to foreseen as follow-up of the work presented in this dissertation are described next, divided accordingly to the three high-level services studied:

- **Security:** securing completely an NoC-based MPSoC is an unlikely feasible and challenging task, due to the possible vulnerabilities brought by increasing new functionalities offered by modern complex multiprocessor platforms, and by the increased connectivity of new embedded platforms, which offers attackers additional means to access remotely the device. While this dissertation proposed possible solutions that may help in this task, still work can be done in the direction of enhancing security in NoC platforms. For instance, future work may involve the analysis of traffic behaviors of possible attacks and the integration of the monitoring system with software strategies to detect possible security threats. An integration of the presented probes with other hardware monitors should also be analyzed, with the aim to implement a hardware/software based Intrusion Detection System for NoCs. The identification of malicious patterns in NoC traffic and operations represents however an open research topic. While some ideas have been proposed in [17, 122, 123], a clear and efficient solution for the problem is still missing - actually the identification of unnatural traffic behaviors is still an open research problem also in the domain of data networks [241, 242]. However, being NoC traffic and operations, with respect to data networks, limited in terms of connections and actors involved in the communication, "intrusion detection" techniques based on profiling of NoC activities could represent an interesting research direction to be explored.

- **NoC monitoring:** the work presented in this dissertation mainly focused on the monitoring of NoC activities. However, in order to have a complete overview of the architecture behavior at system level, data collected by the proposed monitoring system should be considered jointly with monitoring data coming from hardware probes located within processing and storage elements. Future work should therefore go in the direction of providing a comprehensive system-level monitoring system for NoC-based MPSoCs that could help developers in the optimization, both at design time and at run-time, of the usage of system resources.

- **Fault tolerance:** the presented fault tolerant architectures increase fault tolerance of the NI at a relatively low area and energy consumption overhead; however, the main drawback of the proposed techniques is represented by the increased critical path of the designs. Alternative pipelined versions of the architectures can be implemented, increasing however the latency of the communication. As the techniques proposed focus on the basic architectural components of

the NI (LUT, FIFOs, FSMs), we can foresee extension of similar techniques also to NoC routers, composed in general of the same type of elements. Similarly, the methodology applied in section 7.5 for selecting the best architectural configuration for the NI can be generalized and extended to the overall NoC, by providing a way for including fault tolerance - in our specific case the survivability of the system - as optimization parameter to be employed in the early design space exploration phase.

Appendix A

Glossary

- **3GPP**. 3rd Generation Partnership Project.
- **BE**. Best Effort.
- **CAM**. Content Addressable Memory.
- **CMP**. Chip Multiprocessors.
- **DEMA**. Differential Electromagnetic Analysis.
- **DES**. Data Encryption Standard.
- **DFA**. Differential Fault Analysis.
- **DFVS**. Dynamic Frequency and Voltage Scaling.
- **DoS**. Denial-of-Service.
- **DoSP**. Denial-of-Service Probe.
- **DPA**. Differential Power Analysis.
- **DPU**. Data Protection Unit.
- **DRM**. Digital Right Management.
- **DSP**. Digital Signal Processor.
- **EM**. Electromagnetic.
- **EMA**. Electromagnetic Analysis.
- **EXEC**. Execution Time.
- **FIFO**. First Input First Output.

- **FPMAC**. Floating-Point Multiply-Accumulator.
- **FPGA**. Field-Programmable Gate Array.
- **FSM**. Finite State Machine.
- **I2I**. Initiator to Initiator.
- **I2T**. Initiator to Target.
- **IAP**. Illegal Access Probe.
- **IP**. Intellectual Propriety.
- **JTAG**. Join Test Action Group.
- **LSB**. Least Significant Bit.
- **LUT**. Lookup Table.
- **MAF**. Maximal Aggressor Fault.
- **MMS**. Multimedia Messaging Service.
- **MMU**. Memory Management Unit.
- **MPSoC**. Multiprocessor System-on-Chip.
- **MSB**. Most Significant Bit.
- **NI**. Network Interface.
- **NoC**. Network-on-Chip.
- **NSM**. Network Security Manager.
- **NTP**. Network Time Protocol.
- **OCP**. Open Core Protocol.
- **OCP-IP**. Open Core Protocol International Partnership.
- **PE**. Processing Element.
- **PMU**. Probes Management Unit.
- **QoS**. Quality-of-Service.
- **SECDED**. Single Error Correcting and Double Error Detecting.
- **SEMA**. Single Electromagnetic Analysis.

- **SER**. Soft Error Rate.
- **SEU**. Single Event Upset.
- **SMT**. Simultaneous Multithreading.
- **SCM**. Secure Configuration Manager.
- **SoC**. System-on-Chip.
- **SPA**. Simple Power Analysis.
- **T2I**. Target to Initiator.
- **TCAM**. Ternary Content Addressable Memory.
- **TMR**. Triple Modular Redundancy.
- **VLIW**. Very Long Instruction Word.

Appendix B

Author's publications

International Journals

1. O. Derin, E. Diken, and L. Fiorin, "A Middleware Approach to Achieving Fault-tolerant of Kahn Process Networks on Networks on Chips". In International Journal of Reconfigurable Computing, vol. 2011, Article ID 295385, 15 pages, 2011.

2. L. Fiorin, G. Palermo, S. Lukovic, V. Catalano, and C. Silvano, "Secure Memory Accesses on Networks-on-Chip". In IEEE Transactions on Computers, Sept. 2008.

Book Chapters

1. L. Fiorin, G. Palermo, C. Silvano, and M. Sami, "Security in NoC". In "Networks-on-Chips: Theory and Practice", Fayez Gebali and Haytham Elmiligi (Editors), Taylor & Francis Group, LLC, 2009.

International Conferences, Symposia and Workshops

1. P. Meloni, G. Tuveri, L. Raffo, E. Cannella, T. Stefanov, O. Derin, L. Fiorin, and M. Sami, "System Adaptivity and Fault-tolerance in NoC-based MPSoCs: the MADNESS Project Approach". In proceedings of the 15th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD'12). Sept. 5-8, 2012, Izmir, Turkey.

2. L. Fiorin, A. Ferrante, K. Padarnitsas, and F. Regazzoni, "Security Enhanced Linux on Embedded Systems: a Hardware-accelerated Implementation". In proceedings of the 17th Asia and South Pacific Automation Conference (ASP-DAC 2012). Jan. 30 - Feb. 2, 2012, Sydney, Australia.

3. E. Cannella, L. Di Gregorio, L. Fiorin, M. Lindwer, P. Meloni, O. Neugebauer, and A. Pimentel, "Towards an ESL Design Framework for Adaptive and Fault-tolerant MPSoCs: MADNESS or not?". In proceedings of the 9th IEEE/ACM Symposium on Embedded Systems For Real-time Multimedia (ESTIMedia 2011). Oct. 13-14 2011, Taipei, Taiwan.

4. L. Fiorin, L. Micconi, and M. Sami, "Design of Fault Tolerant Network Interfaces for NoCs". In proceedings of 14th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD'11). Aug. 31 - Sept. 2, 2011, Oulu, Finland.

5. O. Derin, D. Kabakci, and L. Fiorin, "Online Task Remapping Strategies for Fault-tolerant Network-on-Chip Multiprocessors Monitoring System for NoCs". In proceedings of the 5th ACM/IEEE International Symposium on Networks-on-Chip (NOCS 2011). May 1-4, 2011, Pittsburgh, Pennsylvania, USA.

6. L. Fiorin, G. Palermo, and C. Silvano, "A Monitoring System for NoCs". In proceedings of the Third International Workshop on Network on Chip Architectures (NoCArc'2010). Dec. 4, 2010, Atlanta, Georgia, USA.

7. L. Fiorin, A. Ferrante, K. Padarnitsas, and S. Carucci, "Hardware-assisted Security Enhanced Linux in Embedded Systems: a Proposal". In proceedings of the 5th Workshop on Embedded Systems Security (WESS'10). Oct. 24, 2010, Scottsdale, Aritzona, USA.

8. S. Lukovic, P. Pezzino, and L. Fiorin, "Stack Protection Unit as a step towards securing MPSoCs". In proceedings of the 24th IEEE International Parallel & Distributed Processing Symposium (IPDPS'10). Apr. 19-23, 2010, Atlanta, Georgia, USA.

9. L. Fiorin, G. Palermo, and C. Silvano, "MPSoCs Run-Time Monitoring through Networks-on-Chip". In proceedings of the 2009 Conference on Design, Automation & Test In Europe (DATE'09), Apr. 20 - 24, 2009, Nice, France.

10. L. Fiorin, G. Palermo, and C. Silvano, "A Security Monitoring Service for NoCs". In proceedings of the Sixth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'08), Oct. 19 - 24, 2008, Atlanta, Georgia, USA.

11. S. Lukovic and L. Fiorin, "An Automated Design Flow for NoC-based MPSoCs on FPGA". In proceedings of the 19th IEEE/IFIP International Symposium on Rapid System Prototyping. June 2-5, 2008, Monterey, CA, USA.

12. L. Fiorin, S. Lukovic, and G. Palermo, "Implementation of a Reconfigurable Data Protection Module for NoC-based MPSoC". In proceedings of the 22nd IEEE

International Parallel & Distributed Processing Symposium (IPDPS'08). April 14-15, 2008, Miami, Florida, USA.

13. L. Fiorin, G. Palermo, S. Lukovic, and C. Silvano, "A Data Protection Unit for NoC-based Architectures". In proceedings of the Fifth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'07), Sept. 30 - Oct. 5, 2007, Salzburg, Austria.

14. L. Fiorin, C. Silvano, and M. Sami, "Security Aspects in Networks-on-Chips: Overview and Proposals for Secure Implementations". In proceedings of 10th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD'07), Aug. 29 - 31 2007, Lübeck, Germany.

Patents

1. V. Catalano, R. Locatelli, M. Coppola, C. Silvano, G. Palermo, L. Fiorin, "Programmable data protection device, secure programming manager system and process for controlling access to an interconnect network for an integrated circuit". US Patent no. US 8,185,934.

2. V. Catalano, R. Locatelli, M. Coppola, C. Silvano, G. Palermo, L. Fiorin, "Programmable data protection device, secure programming manager system and process for controlling access to an interconnect network for an integrated circuit". European Patent Application no. 07301411.0 - 2413.

Other publications

1. O. Derin, L. Fiorin, M. Sami, P. Meloni, S. Secchi, and L. Raffo , "Fault-tolerance of Kahn Process Networks on NoC-based heterogeneous multicore embedded architectures". Presented at Intel European Research and Innovation Conference (ERIC). Sept. 21-22, 2010, Braunschweig, Germany.

Submitted for publication

1. P. Meloni, G. Tuveri, E. Cannella, O. Derin, T. Stefanov, L. Fiorin, L. Raffo, and M. Sami, "A System-level Approach to Adaptivity and Fault-tolerance in NoC-based MPSoCs: the MADNESS Project". *Submitted to* Elsevier Microprocessors and Microsystems (MICPRO).

2. L. Fiorin, and G. Palermo, and C. Silvano, "A Configurable Monitoring Infrastructure for NoC-based Architectures". *Submitted to* IEEE Transactions on Very Large Scale Integration (VLSI) Systems.

3. L. Fiorin, and M. Sami, "Online Fault Detection and Reconfiguration in Fault Tolerant Network Interfaces for NoCs". *Submitted to* IEEE Transactions on Dependable and Secure Computing.

Bibliography

[1] F. Regazzoni, S. Badel, T. Eisenbarth, J. Grobschadl, A. Poschmann, Z. Toprak, M. Macchetti, L. Pozzi, C. Paar, Y. Leblebici, and P. Ienne, "A Simulation-Based Methodology for Evaluating the DPA-Resistance of Cryptographic Functional Units with Application to CMOS and MCML Technologies," in *Embedded Computer Systems: Architectures, Modeling and Simulation, 2007. IC-SAMOS 2007. International Conference on*, july 2007, pp. 209 –214.

[2] ITRS, "Itrs 2010 documents, http://www.itrs.net/links/2010itrs/home2010.htm."

[3] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, N. Belfiori, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 1, pp. 29 –41, jan. 2008.

[4] J. Howard, S. Dighe, S. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, and R. Van Der Wijngaart, "A 48-core IA-32 Processor in 45 nm CMOS Using On-Die Message-Passing and DVFS for Performance and Power Scaling," *Solid-State Circuits, IEEE Journal of*, vol. 46, no. 1, pp. 173 –183, jan. 2011.

[5] G. D. Micheli and L. Benini, *Networks on Chips: Technology and Tools (Systems on Silicon)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

[6] W. J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Inteconnection Networks," in *Proceedings of the 38th annual Design Automation Conference*, ser. DAC '01. New York, NY, USA: ACM, 2001, pp. 684–689. [Online]. Available: http://doi.acm.org/10.1145/378239.379048

[7] L. Benini and G. De Micheli, "Networks on Chips: A New SoC Paradigm," *Computer*, vol. 35, no. 1, pp. 70 –78, jan 2002.

[8] *Open Core Protocol Specification 2.2*.

[9] A. Radulescu, J.], S. Pestana, O. Gangwal, E. Rijpkema, P. Wielage, and K. Goossens, "An Efficient On-Chip NI Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, no. 1, pp. 4 – 17, jan. 2005.

[10] I. Loi, F. Angiolini, and L. Benini, "Synthesis of Low-Overhead Configurable Source Routing Tables for Network Interfaces," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, april 2009, pp. 262 –267.

[11] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, "Security in Embedded Systems: Design Challenges," *ACM Trans. Embed. Comput. Syst.*, vol. 3, no. 3, pp. 461–491, Aug. 2004. [Online]. Available: http://doi.acm.org/10.1145/1015047.1015049

[12] P. Kocher, R. Lee, G. McGraw, A. Raghunathan, and S. Ravi, "Security As A New Dimension in Embedded System Design," in *Design Automation Conference, 2004. Proceedings. 41st*, july 2004, pp. 753 –760.

[13] R. Vaslin, G. Gogniat, and J. P. Diguet, "Secure Architecture in Embedded Systems: an Overview," in *Workshop on Reconfigurable Communication-Centric SoCs (ReCoSoC'06)*, July 3-5 2006.

[14] D. Nash, T. Martin, D. Ha, and M. Hsiao, "Towards an Intrusion Detection System for Battery Exhaustion Attacks on Mobile Computing Devices," in *Pervasive Computing and Communications Workshops, 2005. PerCom 2005 Workshops. Third IEEE International Conference on*, march 2005, pp. 141 – 145.

[15] "SymbOS.Cabir," Symantec Corporation, Tech. Rep., 2004.

[16] J. Niemela, *F-Secure Virus Descriptions*, F-Secure, December 2007. [Online]. Available: http://www.f-secure.com/v-descs/worm_symbos_beselo.shtml

[17] L. Fiorin, C. Silvano, and M. Sami, "Security Aspects in Networks-on-Chips: Overview and Proposals for Secure Implementations," in *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*, aug. 2007, pp. 539 –542.

[18] J.-P. Diguet, S. Evain, R. Vaslin, G. Gogniat, and E. Juin, "NOC-Centric Security of Reconfigurable SoC," in *Networks-on-Chip, 2007. NOCS 2007. First International Symposium on*, may 2007, pp. 223 –232.

[19] J. Coburn, S. Ravi, A. Raghunathan, and S. Chakradhar, "SECA: Security-Enhanced Communication Architecture," in *Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*,

ser. CASES '05. New York, NY, USA: ACM, 2005, pp. 78–89. [Online]. Available: http://doi.acm.org/10.1145/1086297.1086308

[20] S. Ravi, A. Raghunathan, and S. Chakradhar, "Tamper Resistance Mechanisms for Secure Embedded Systems," in *VLSI Design, 2004. Proceedings. 17th International Conference on*, 2004, pp. 605 – 611.

[21] E. Chien and P. Szoe, *Blended Attacks Exploits, Vulnerabilities and Buffer Overflow Techniques in Computer Viruses*, Symantec White Paper, Sept. 2002.

[22] J. Niemela, *Skulls.D - Virus Descriptions*, F-Secure, October 2005. [Online]. Available: http://www.f-secure.com/v-descs/skulls_d.shtml

[23] F. Koeune and F. X. Standaerd, *Foundations of Security Analysis and Design III*. Springer Berlin / Heidelberg, 2005, vol. 3655/2005.

[24] P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '99. London, UK, UK: Springer-Verlag, 1999, pp. 388–397. [Online]. Available: http://dl.acm.org/citation.cfm?id=646764.703989

[25] "35.202 Technical Specification Version 3.1.1. Kasumi S-box Function Specifications," 3GPP, Tech. Rep., 2002, http: //www.3gpp. org/ftp/Specs/archive/35-series/35.202/,.

[26] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the Importance of Eliminating Errors in Cryptographic Computations," *J. Cryptology*, vol. 14, pp. 101 – 119, December 2001.

[27] E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability," in *Proceedings of the 14th Design Automation Conference*, ser. DAC '77. Piscataway, NJ, USA: IEEE Press, 1977, pp. 462–468. [Online]. Available: http://dl.acm.org/citation.cfm?id=800262.809170

[28] B. Yang, K. Wu, and R. Karri, "Scan Based Side Channel Attack on Dedicated Hardware Implementations of Data Encryption Standard," in *Test Conference, 2004. Proceedings. ITC 2004. International*, oct. 2004, pp. 339 – 344.

[29] S. Evain and J.-P. Diguet, "From NoC Security Analysis to Design Solutions," in *Signal Processing Systems Design and Implementation, 2005. IEEE Workshop on*, nov. 2005, pp. 166 – 171.

[30] T. Martin, M. Hsiao, D. Ha, and J. Krishnaswami, "Denial-of-Service Attacks on Battery-Powered Mobile Computers," in *Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on*, march 2004, pp. 309 – 318.

[31] T. Simunic, S. Boyd, and P. Glynn, "Managing Power Consumption in Networks on Chips," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 12, no. 1, pp. 96 –107, jan. 2004.

[32] *Digital Audio over IEEE1394, White Paper*, Oxford Semiconductor, Jan. 2003.

[33] Y. Zhang, L. Gao, J. Yang, X. Zhang, and R. Gupta, "Senss: Security Enhancement to Symmetric Shared Memory Multiprocessors," in *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, feb. 2005, pp. 352 – 362.

[34] B. Rogers, M. Prvulovic, and Y. Solihin, "Efficient Data Protection for Distributed Shared Memory Multiprocessors," in *Proceedings of the 15th international conference on Parallel architectures and compilation techniques*, ser. PACT '06. New York, NY, USA: ACM, 2006, pp. 84–94. [Online]. Available: http://doi.acm.org/10.1145/1152154.1152170

[35] "www.modchip.com."

[36] S. Hansman and R. Hunt, "A Taxonomy of Network and Computer Attacks." *Computers & Security*, vol. 24, no. 1, pp. 31–43, 2005.

[37] "US-CERT Vulnerability Notes Database." US CERT, Tech. Rep., 2007. [Online]. Available: http://www.kb.cert.org/vuls

[38] D. H. Woo and H. S. Lee, "Analyzing Performance Vulnerability due to Resource Denial-of-Service Attack on Chip Multiprocessors," in *Proc. of ISCA'07,* June 9-13 2007.

[39] D. Grunwald and S. Ghiasi, "Microarchitectural Denial of Service: Insuring Microarchitectural Fairness," in *Microarchitecture, 2002. (MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on*, 2002, pp. 409 – 418.

[40] J. Hasan, A. Jalote, T. Vijaykumar, and C. Brodley, "Heat Stroke: Power-Density-Based Denial of Service in SMT," in *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, feb. 2005, pp. 166 – 177.

[41] ARM, "Technical Information," http://www.arm.com/products/processors/classic/arm9/index.php.

[42] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "Mibench: A Free, Commercially Representative Embedded Benchmark Suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, dec. 2001, pp. 3 – 14.

[43] "Simplescalar/arm," http://www.simplescalar.com/v4test.html.

[44] B. Sprunt, "Pentium 4 Performance-monitoring Features," *Micro, IEEE*, vol. 22, no. 4, pp. 72 – 82, jul/aug 2002.

[45] V. Nollet, D. Verkest, and H. Corporaal, "A Safari Through the MPSoC Run-Time Management Jungle," vol. 60, no. 2. Hingham, MA, USA: Kluwer Academic Publishers, Aug. 2010, pp. 251–268. [Online]. Available: http://dx.doi.org/10.1007/s11265-008-0305-4

[46] "IST-027611-AETHER-Self-Adaptive Embedded Technologies for Pervasive Computing Architectures." [Online]. Available: http://www.aether-ist.org/

[47] "IST-4-027342-MORPHEUS - Multi-purpOse dynamically Reconfigurable Platform for intensive HEterogeneoUS processing." [Online]. Available: http://www.morpheus-ist.org/

[48] A. Bartolini, M. Cacciari, A. Tilli, L. Benini, and M. Gries, "A virtual platform environment for exploring power, thermal and reliability management control strategies in high-performance multicores," in *Proceedings of the 20th symposium on Great lakes symposium on VLSI*, ser. GLSVLSI '10. New York, NY, USA: ACM, 2010, pp. 311–316. [Online]. Available: http://doi.acm.org/10.1145/1785481.1785553

[49] S. Hong, S. Yoo, S. Lee, S. Lee, H. J. Nam, B.-S. Yoo, J. Hwang, D. Song, J. Kim, J. Kim, H. Jin, K.-M. Choi, J.-T. Kong, and S. Eo, "Creation and utilization of a virtual platform for embedded software optimization:: an industrial case study," in *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*, ser. CODES+ISSS '06. New York, NY, USA: ACM, 2006, pp. 235–240. [Online]. Available: http://doi.acm.org/10.1145/1176254.1176311

[50] D. Mangano and G. Strano, "Enabling Dynamic and Programmable QoS in SoCs," in *Proceedings of the Third International Workshop on Network on Chip Architectures*, ser. NoCArc '10. New York, NY, USA: ACM, 2010, pp. 17–22. [Online]. Available: http://doi.acm.org/10.1145/1921249.1921255

[51] J. Diemer and R. Ernst, "Back Suction: Service Guarantees for Latency-Sensitive on-Chip Networks," in *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*, may 2010, pp. 155 –162.

[52] N. Muralimanohar and R. Balasubramonian, "Interconnect Design Considerations for Large NUCA Caches," in *Proceedings of the 34th annual international symposium on Computer architecture*, ser. ISCA '07. New York, NY, USA: ACM, 2007, pp. 369–380. [Online]. Available: http://doi.acm.org/10.1145/1250662.1250708

[53] V. Nollet, T. Marescaux, D. Verkest, J.-Y. Mignolet, and S. Vernalde, "Operating-System Controlled Network on Chip," in *Proceedings of the 41st annual Design Automation Conference*, ser. DAC '04. New York, NY, USA: ACM, 2004, pp. 256–259. [Online]. Available: http://doi.acm.org/10.1145/996566.996637

[54] L. Fiorin, G. Palermo, S. Lukovic, V. Catalano, and C. Silvano, "Secure Memory Accesses on Networks-on-Chip," *Computers, IEEE Transactions on*, vol. 57, no. 9, pp. 1216 –1229, sept. 2008.

[55] C. E. McDowell and D. P. Helmbold, "Debugging Concurrent Programs," *ACM Comput. Surv.*, vol. 21, pp. 593–622, December 1989. [Online]. Available: http://doi.acm.org/10.1145/76894.76897

[56] M. El Shobaki, "On-Chip Monitoring of Single- and Multiprocessor Hardware Real-Time Operating Systems," in *Proc. of RTCSA'02*, March, 18 - 20 2002.

[57] A.-C. Liu and R. Parthasarathi, "Hardware Monitoring of A Multiprocessor System," *Micro, IEEE*, vol. 9, no. 5, pp. 44 –51, oct. 1989.

[58] F. Jahanian, R. Rajkumar, and S. C. V. Raju, "Runtime Monitoring of Timing Constraints in Distributed Real-ime Systems," *Real-Time Syst.*, vol. 7, no. 3, pp. 247–273, Nov. 1994. [Online]. Available: http://dx.doi.org/10.1007/BF01088521

[59] R. Marculescu, "Networks-on-Chip: The Quest for on-Chip Fault-Tolerant Communication," in *VLSI, 2003. Proceedings. IEEE Computer Society Annual Symposium on*, feb. 2003, pp. 8 – 12.

[60] J. Srinivasan and S. V. Adve, "IBM Research Report RAMP : A Model for Reliability Aware MicroProcessor Design," *Rivers*, vol. 23048, 2003.

[61] I. Koren and C. M. Krishna, *Fault Tolerant Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[62] A. Ferrante, S. Medardoni, and D. Bertozzi, "Network Interface Sharing Techniques for Area Optimized NoC Architectures," in *Digital System Design Architectures, Methods and Tools, 2008. DSD '08. 11th EUROMICRO Conference on*, sept. 2008, pp. 10 –17.

[63] P. Roche and G. Gasiot, "Impacts of Front-end and Middle-end Process Modifications on Terrestrial Soft Error Rate," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, no. 3, pp. 382 – 396, sept. 2005.

[64] D. Zhu, R. Melhem, and D. Mosse, "The Effects of Energy Management on Reliability in Real-time Embedded Systems," in *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, nov. 2004, pp. 35 – 40.

[65] "Tezzaron Semiconductor. Soft Errors in Electronic Memory: A White Paper," 2004. [Online]. Available: http://tezzaron.com/about/papers/soft_errors_1_1_secure.pdf

[66] J. B. Bernstein, M. Gurfinkel, X. Li, J. Walters, Y. Shapira, and M. Talmor, "Electronic Circuit Reliability Modeling," *Microelectronic Reliability*, no. 46, pp. 1957–1979, 2006.

[67] "Nangate 45nm Open Cell Library." [Online]. Available: http://www.nangate.com/

[68] T. Bengtsson, S. Kumar, and Z. Peng, "Application Area Specific System Level Fault Models: A Case Study with A Simple NoC Switch," in *In Proceedings (electronic) of International Design and Test Workshop (IDT)*, 2006.

[69] A. Frantz, M. Cassel, F. Kastensmidt, E. Cota, and L. Carro, "Crosstalk- and SEU-Aware Networks on Chips," *Design Test of Computers, IEEE*, vol. 24, no. 4, pp. 340 –350, july-aug. 2007.

[70] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures," *Computers, IEEE Transactions on*, vol. 54, no. 8, pp. 1025 – 1040, aug. 2005.

[71] K. Goossens, J. Dielissen, and A. Radulescu, "AEthereal Network on Chip: Concepts, Architectures, and Implementations," *Design Test of Computers, IEEE*, vol. 22, no. 5, pp. 414 – 421, sept.-oct. 2005.

[72] A. Leroy, P. Marchal, A. Shickova, F. Catthoor, F. Robert, and D. Verkest, "Spatial Division Multiplexing: A Novel Approach for Guaranteed Throughput on NoCs," in *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/-software codesign and system synthesis*, ser. CODES+ISSS '05. New York, NY, USA: ACM, 2005, pp. 81–86.

[73] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed Bandwidth using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 2, feb. 2004, pp. 890 – 895 Vol.2.

[74] M. Coppola, S. Curaba, M. Grammatikakis, G. Maruccia, and F. Papariello, "OCCN: A Network-on-Chip Modeling and Simulation Framework," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 3, feb. 2004, pp. 174 – 179 Vol.3.

[75] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks - An Engineering Approach*. Morgan Kaufmann Publishers Inc., 2002.

[76] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS Architecture and Design Process for Network on Chip," *J. Syst. Archit.*, vol. 50, no. 2-3, pp. 105–128, Feb. 2004. [Online]. Available: http://dx.doi.org/10.1016/j.sysarc.2003.07.004

[77] A. Guerre, N. Ventroux, R. David, and A. Merigot, "Hierarchical Network-on-Chip for Embedded Many-Core Architectures," in *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*, may 2010, pp. 189 –196.

[78] U. Y. Ogras, J. Hu, and R. Marculescu, "Key Research Problems in NoC Design: A Holistic Perspective," in *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ser. CODES+ISSS '05. New York, NY, USA: ACM, 2005, pp. 69–74.

[79] T. Bjerregaard and S. Mahadevan, "A Survey of Research and Practices of Network-on-Chip," *ACM Comput. Surv.*, vol. 38, no. 1, Jun. 2006.

[80] M. Palesi, R. Holsmark, and S. Kumar, "A Methodology for Design of Application Specific Deadlock-free Routing Algorithms for NoC Systems," in *Hardware/Software Codesign and System Synthesis, 2006. CODES+ISSS '06. Proceedings of the 4th International Conference*, oct. 2006, pp. 142 –147.

[81] J. Hu and R. Marculescu, "Dyad: Smart Routing for Networks-on-Chip," in *Proceedings of the 41st annual Design Automation Conference*, ser. DAC '04. New York, NY, USA: ACM, 2004, pp. 260–263. [Online]. Available: http://doi.acm.org/10.1145/996566.996638

[82] K. Srinivasan and K. Chatha, "A Technique for Low Energy Mapping and Routing in Network-on-Chip Architectures," in *Low Power Electronics and Design, 2005. ISLPED '05. Proceedings of the 2005 International Symposium on*, aug. 2005, pp. 387 – 392.

[83] N. Concer, S. Iamundo, and L. Bononi, "aEqualized: A Novel Routing Algorithm for The Spidergon Network On Chip," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, april 2009, pp. 749 –754.

[84] A. Sharifi and M. Kandemir, "Process Variation-Aware Routing in NoC Based Multicores," in *Proceedings of the 48th Design Automation Conference*, ser. DAC '11. New York, NY, USA: ACM, 2011, pp. 924–929. [Online]. Available: http://doi.acm.org/10.1145/2024724.2024930

[85] J. Cong, C. Liu, and G. Reinman, "ACES: Application-Specific Cycle Elimination and Splitting for Deadlock-free Routing on Irregular Network-on-Chip," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10. New York, NY, USA: ACM, 2010, pp. 443–448. [Online]. Available: http://doi.acm.org/10.1145/1837274.1837385

[86] J. Cano, J. Flich, J. Duato, M. Coppola, and R. Locatelli, "Efficient Routing Implementation in Complex Systems-on-Chip," in *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*, may 2011, pp. 1 –8.

[87] U. Y. Ogras and R. Marculescu, "Prediction-Based Flow Control for Network-on-Chip Traffic," in *Proceedings of the 43rd annual Design Automation Conference*, ser. DAC '06. New York, NY, USA: ACM, 2006, pp. 839–844.

[88] Y. J. Yoon, N. Concer, M. Petracca, and L. Carloni, "Virtual Channels vs. Multiple Physical Networks: a Comparative Analysis," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10. New York, NY, USA: ACM, 2010, pp. 162–165. [Online]. Available: http://doi.acm.org/10.1145/1837274.1837315

[89] A. Joshi and M. Mutyam, "Prevention Flow-Control for Low Latency Torus Networks-on-Chip," in *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*, may 2011, pp. 41 –48.

[90] J. Hu and R. Marculescu, "Application-Specific Buffer Space Allocation for Networks-on-Chip Router Design," in *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, nov. 2004, pp. 354 – 361.

[91] G. Kim, J. Kim, and S. Yoo, "FlexiBuffer: Reducing Leakage Power in on-Chip Network Routers," in *Proceedings of the 48th Design Automation Conference*, ser. DAC '11. New York, NY, USA: ACM, 2011, pp. 936–941. [Online]. Available: http://doi.acm.org/10.1145/2024724.2024932

[92] T. Ono and M. Greenstreet, "A Modular Synchronizing FIFO for NoCs," in *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, may 2009, pp. 224 –233.

[93] Z. Guz, I. Walter, E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Efficient Link Capacity and QoS Design for Network-on-Chip," in *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, vol. 1, march 2006, pp. 1 –6.

[94] E. Rijpkema, K. G. W. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander, "Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip," in *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*, ser. DATE '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 10 350–. [Online]. Available: http://dl.acm.org/citation.cfm?id=789083.1022751

[95] M. A. Al Faruque, G. Weiss, and J. Henkel, "Bounded Arbitration Algorithm for QoS-Supported on-Chip Communication," in *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*, ser. CODES+ISSS '06. New York, NY, USA: ACM, 2006, pp. 76–81. [Online]. Available: http://doi.acm.org/10.1145/1176254.1176275

[96] M. Harmanci, N. Escudero, Y. Leblebici, and P. Ienne, "Quantitative Modelling and Comparison of Communication Schemes to Guarantee Quality-of-Service in Networks-on-Chip," in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, may 2005, pp. 1782 – 1785 Vol. 2.

[97] A. Sharifi, H. Zhao, and M. Kandemir, "Feedback Control for Providing QoS in NoC Based Multicores," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, march 2010, pp. 1384 –1389.

[98] A. Hansson, M. Subburaman, and K. Goossens, "Aelite: A Flit-Synchronous Network on Chip with Composable and Predictable Services," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, april 2009, pp. 250 –255.

[99] D. Ludovici, A. Strano, G. Gaydadjiev, L. Benini, and D. Bertozzi, "Design Space Exploration of A Mesochronous Link for Cost-Effective and Flexible GALS NoCs," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, march 2010, pp. 679 –684.

[100] D. Gebhardt, J. You, and K. Stevens, "Link Pipelining Strategies for an Application-Specific Asynchronous NoC," in *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*, may 2011, pp. 185 –192.

[101] C. HernaÌĄndez, A. Roca, F. Silla, J. Flich, and J. Duato, "Improving the Performance of GALS-Based NoCs in The Presence of Process Variation," in *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*, may 2010, pp. 35 –42.

[102] D. Gebhardt, J. You, and K. Stevens, "Comparing Energy and Latency of Asynchronous and Synchronous NoCs for Embedded SoCs," in *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*, may 2010, pp. 115 –122.

[103] H. Gu, J. Xu, and W. Zhang, "A Low-power Fat Tree-based Optical Network-On-Chip for Multiprocessor System-On-Chip," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, april 2009, pp. 3 –8.

[104] X. Zhang and A. Louri, "A Multilayer Nanophotonic Interconnection Network for on-Chip Many-Core Communications," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10. New York, NY, USA: ACM, 2010, pp. 156–161. [Online]. Available: http://doi.acm.org/10.1145/1837274.1837314

[105] Y. Xie, M. Nikdast, J. Xu, W. Zhang, Q. Li, X. Wu, Y. Ye, X. Wang, and W. Liu, "Crosstalk Noise and Bit Error Rate Analysis for Optical Network-on-Chip," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10. New York, NY, USA: ACM, 2010, pp. 657–660. [Online]. Available: http://doi.acm.org/10.1145/1837274.1837441

[106] A. Abousamra, R. Melhem, and A. Jones, "Two-Hop Free-Space Based Optical Interconnects for Chip Multiprocessors," in *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*, may 2011, pp. 89 –96.

[107] A. Kodi, R. Morris, A. Louri, and X. Zhang, "On-Chip Photonic Interconnects for Scalable Multi-Core Architectures," in *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, may 2009, p. 90.

[108] D. Zhao, Y. Wang, J. Li, and T. Kikkawa, "Design of Multi-Channel Wireless NoC to Improve on-Chip Communication Capacity," in *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*, may 2011, pp. 177 –184.

[109] C. Seiculescu, S. Murali, L. Benini, and G. De Micheli, "SunFloor 3D: A Tool for Networks On Chip Topology Synthesis for 3D Systems on Chips," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, april 2009, pp. 9 –14.

[110] M. Ebrahimi, M. Daneshtalab, P. Liljeberg, J. Plosila, and H. Tenhunen, "Exploring Partitioning Methods for 3D Networks-on-Chip Utilizing Adaptive Routing Model," in *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*, may 2011, pp. 73 –80.

[111] A. Weldezion, M. Grange, D. Pamunuwa, Z. Lu, A. Jantsch, R. Weerasekera, and H. Tenhunen, "Scalability of network-on-Chip Communication Architecture for 3-D Meshes," in *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, may 2009, pp. 114 –123.

[112] *XOM Technical Information*, http://www-vlsi.stanford.edu/ lie/xom.htm.

[113] G. Suh, C. O'Donnell, I. Sachdev, and S. Devadas, "Design and Implementation of The AEGIS Single-Chip Secure Processor Using Physical Random Functions," in *Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on*, june 2005, pp. 25 – 36.

[114] C. Gebotys and R. Gebotys, "A Framework for Security on NoC Technologies," in *VLSI, 2003. Proceedings. IEEE Computer Society Annual Symposium on*, feb. 2003, pp. 113 – 117.

[115] C. H. Gebotys and Y. Zhang, "Security Wrappers and Power Analysis for SoC Technologies," in *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ser. CODES+ISSS '03. New York, NY, USA: ACM, 2003, pp. 162–167. [Online]. Available: http://doi.acm.org/10.1145/944645.944689

[116] T. Alves and D. Felton, *TrustZone: Integrated Hardware and Software Security, White Paper*, ARM, 2004.

[117] *SonicsMX SMART Interconnect Datasheet*, http://www.sonicsinc.com.

[118] H. kai Li, J. Han, X. yang Zeng, and S. yu Zhang, "Implementation of A Reconfigurable Data Protection Controller on NoC System," in *Solid-State and Integrated Circuit Technology (ICSICT), 2010 10th IEEE International Conference on*, nov. 2010, pp. 485 –487.

[119] J. Porquet, A. Greiner, and C. Schwarz, "NoC-MPU: A Secure Architecture for Flexible co-hosting on Shared Memory MPSoCs," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, march 2011, pp. 1 –4.

[120] P. Cotret, J. Crenne, G. Gogniat, J. Diguet, L. Gaspar, and G. Duc, "Distributed Security for Communications and Memories in a Multiprocessor Architecture," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, may 2011, pp. 326 –329.

[121] J. Sepúlveda, R. Pires, M. Strum, and W. J. Chau, "Transactions on Computational Science X," M. L. Gavrilova, C. J. K. Tan, and E. D. Moreno, Eds. Berlin, Heidelberg: Springer-Verlag, 2010, ch. Implementation of QoSS (quality-of-security service) for NoC-based SoC protection, pp. 187–201. [Online]. Available: http://dl.acm.org/citation.cfm?id=1985581.1985589

[122] A. Mandal, S. K. Mandal, A. Tripathy, N. Gupta, and R. Mahapatra, "A Bio-Inspired Framework for Secure System on Chip," in *Proceedings of Workshop on SoC Architecture, Accelerators and Workloards (SAW-I)*, Jan. 10 2010.

[123] S. Lukovic and N. Christianos, "Hierarchical Multi-agent Protection System for NoC based MPSoCs," in *Proceedings of the International Workshop on Security and Dependability for Resource Constrained Embedded Systems*, ser. S&D4RCES '10. New York, NY, USA: ACM, 2010, pp. 6:1–6:7. [Online]. Available: http://doi.acm.org/10.1145/1868433.1868441

[124] K. Sajeesh and H. Kapoor, "An Authenticated Encryption Based Security Framework for NoC Architectures," in *Electronic System Design (ISED), 2011 International Symposium on*, dec. 2011, pp. 134 –139.

[125] M. Mirza-Aghatabar and A. Sadeghi, "An Asynchronous, Low Power and Secure Framework for Network-On-Chips," *International Journal of Computer Science and Network Security*, vol. 8, no. 7, pp. 214–223, 2008.

[126] R. Stefan and K. Goossens, "Enhancing the Security of Time-Division-Multiplexing Networks-on-Chip Through the Use of Multipath Routing," in *Proceedings of the 4th International Workshop on Network on Chip Architectures*, ser. NoCArc '11. New York, NY, USA: ACM, 2011, pp. 57–62. [Online]. Available: http://doi.acm.org/10.1145/2076501.2076513

[127] B. Vermeulen and K. Goossens, "Interactive Debug of SoCs with Multiple Clocks," *Design Test of Computers, IEEE*, vol. 28, no. 3, pp. 44 –51, may-june 2011.

[128] H. Yi, S. Park, and S. Kundu, "On-Chip Support for NoC-Based SoC Debugging," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 57, no. 7, pp. 1608 –1617, july 2010.

[129] C. Ciordas, T. Basten, A. Rădulescu, K. Goossens, and J. V. Meerbergen, "An Event-Based Monitoring Service for Networks on Chip," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 10, no. 4, pp. 702–723, Oct. 2005. [Online]. Available: http://doi.acm.org/10.1145/1109118.1109126

[130] S. Tang and Q. Xu, "A Multi-Core Debug Platform for NoC-Based Systems," in *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, april 2007, pp. 1 –6.

[131] K. Goossens, B. Vermeulen, R. van Steeden, and M. Bennebroek, "Transaction-Based Communication-Centric Debug," in *Networks-on-Chip, 2007. NOCS 2007. First International Symposium on*, may 2007, pp. 95 –106.

[132] B. Vermeulen and K. Goossens, "A Network-on-Chip Monitoring Infrastructure for Communication-Centric Debug of Embedded Multi-Processor SoCs," in *VLSI Design, Automation and Test, 2009. VLSI-DAT '09. International Symposium on*, april 2009, pp. 183 –186.

[133] J. van den Brand, C. Ciordas, K. Goossens, and T. Basten, "Congestion-Controlled Best-Effort Communication for Networks-on-Chip," in *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, april 2007, pp. 1 –6.

[134] V. Nollet, P. Avasare, H. Eeckhaut, D. Verkest, and H. Corporaal, "Run-Time Management of a MPSoC Containing FPGA Fabric Tiles," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 1, pp. 24 –33, jan. 2008.

[135] M. Al Faruque, T. Ebi, and J. Henkel, "ROAdNoC: Runtime Observability for An Adaptive Network on Chip Architecture," in *Computer-Aided Design, 2008. ICCAD 2008. IEEE/ACM International Conference on*, nov. 2008, pp. 543 –548.

[136] B. Vermeulen, K. Goossens, and S. Umrani, "Debugging Distributed-Shared-Memory Communication at Multiple Granularities in Networks on Chip," in *Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on*, april 2008, pp. 3 –12.

[137] *Arteris - The Network-on-Chip Company*, http://www.arteris.com.

[138] Z. Wang, "Conception et Analyse Comparative Multi-objectif Multi-Technologies de Famille de Multiprocesseurs sur Puce," Ph.D. dissertation, L'Institut polytechnique de Grenoble, 2009.

[139] P. G. Paulin, C. Pilkington, M. Langevin, E. Bensoudane, and G. Nicolescu, "Parallel Programming Models for a Multi-Processor SoC Platform Applied to High-Speed Traffic Management," in *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ser. CODES+ISSS '04. New York, NY, USA: ACM, 2004, pp. 48–53. [Online]. Available: http://doi.acm.org/10.1145/1016720.1016735

[140] V. Nollet, T. Marescaux, P. Avasare, D. Verkest, and J.-Y. Mignolet, "Centralized Run-Time Resource Management in a Network-on-Chip Containing Reconfigurable Hardware Tiles," in *Design, Automation and Test in Europe, 2005. Proceedings*, march 2005, pp. 234 – 239 Vol. 1.

[141] T. Marescaux, B. Bricke, P. Debacker, V. Nollet, and H. Corporaal, "Dynamic Time-Slot Allocation for QoS Enabled Networks on Chip," in *Embedded Systems for Real-Time Multimedia, 2005. 3rd Workshop on*, sept. 2005, pp. 47 – 52.

[142] A. Kumar, B. Mesman, B. Theelen, H. Corporaal, and H. Yajun, "Resource Manager for Non-Preemptive Heterogeneous Multiprocessor System-on-Chip," in *Embedded Systems for Real Time Multimedia, Proceedings of the 2006 IEEE/ACM/IFIP Workshop on*, oct. 2006, pp. 33 –38.

[143] C. Grecu, "Test and Fault-Tolerance for Network-on-Chip Infrastructures," Ph.D. dissertation, University of British Columbia, 2009.

[144] P. J. Clarke, A. K. Ray, and C. A. Hogarth, "ElectromigrationâĂŤA Tutorial Introduction," *International Journal of Electronics*, vol. 69, no. 3, pp. 333–338, 1990.

[145] M. Cuviello, S. Dey, X. Bai, and Y. Zhao, "Fault Modeling and Simulation for Crosstalk in System-on-Chip Interconnects," in *Computer-Aided Design, 1999. Digest of Technical Papers. 1999 IEEE/ACM International Conference on*, 1999, pp. 297 –303.

[146] H. Zimmer and A. Jantsch, "A Fault Model Notation and Error-Control Scheme for Switch-to-Switch Buses in a Network-on-Chip," in *Hardware/Software Codesign and System Synthesis, 2003. First IEEE/ACM/IFIP International Conference on*, oct. 2003, pp. 188 –193.

[147] E. Cota, F. Kastensmidt, M. Cassel, M. Herve, P. Almeida, P. Meirelles, A. Amory, and M. Lubaszewski, "A High-Fault-Coverage Approach for The Test of Data, Control and Handshake Interconnects in Mesh Networks-on-Chip," *Computers, IEEE Transactions on*, vol. 57, no. 9, pp. 1202 –1215, sept. 2008.

[148] A. Pullini, F. Angiolini, D. Bertozzi, and L. Benini, "Fault Tolerance Overhead in Network-on-Chip Flow Control Schemes," in *Integrated Circuits and Systems Design, 18th Symposium on*, sept. 2005, pp. 224 –229.

[149] S. Murali, T. Theocharides, N. Vijaykrishnan, M. Irwin, L. Benini, and G. De Micheli, "Analysis of Error Recovery Schemes for Networks on Chips," *Design Test of Computers, IEEE*, vol. 22, no. 5, pp. 434 – 442, sept.-oct. 2005.

[150] T. Lehtonen, P. Liljeberg, and J. Plosila, "Online Reconfigurable Self-Timed Links for Fault Tolerant NoC," *VLSI Design*, vol. 2007, p. 13, 2007.

[151] Q. Yu and P. Ampadu, "Transient and Permanent Error Co-management Method for Reliable Networks-on-Chip," in *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*, may 2010, pp. 145 –154.

[152] A. Ejlali, B. Al-Hashimi, P. Rosinger, S. Miremadi, and L. Benini, "Performability/Energy Tradeoff in Error-Control Schemes for on-Chip Networks," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 1, pp. 1 –14, jan. 2010.

[153] K. Stewart and S. Tragoudas, "Interconnect Testing for Networks on Chips," in *VLSI Test Symposium, 2006. Proceedings. 24th IEEE*, april-4 may 2006, p. 6 pp.

[154] J. Kim, C. Nicopoulos, D. Park, V. Narayanan, M. Yousif, and C. Das, "A Gracefully Degrading and Energy-Efficient Modular Router Architecture for on-Chip Networks," in *Computer Architecture, 2006. ISCA '06. 33rd International Symposium on*, 0-0 2006, pp. 4 –15.

[155] A. Alaghi, N. Karimi, M. Sedghi, and Z. Navabi, "Online NoC Switch Fault Detection and Diagnosis Using a High Level Fault Model," in *Defect and Fault-Tolerance in VLSI Systems, 2007. DFT '07. 22nd IEEE International Symposium on*, sept. 2007, pp. 21 –29.

[156] S. Mediratta and J. Draper, "Characterization of a Fault-tolerant NoC Router," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, may 2007, pp. 381 –384.

[157] Y. H. Kang, T.-J. Kwon, and J. Draper, "Fault-Tolerant Flow Control in On-chip Networks," in *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*, may 2010, pp. 79 –86.

[158] D. Fick, A. DeOrio, J. Hu, V. Bertacco, D. Blaauw, and D. Sylvester, "Vicis: A Reliable Network for Unreliable Silicon," in *Proceedings of the 46th Annual Design Automation Conference*, ser. DAC '09. New York, NY, USA: ACM, 2009, pp. 812–817. [Online]. Available: http://doi.acm.org/10.1145/1629911.1630119

[159] M. Koibuchi, H. Matsutani, H. Amano, and T. Mark Pinkston, "A Lightweight Fault-Tolerant Mechanism for Network-on-Chip," in *Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on*, april 2008, pp. 13 –22.

[160] S. Rodrigo, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, J. Camacho, F. Silla, and J. Duato, "Addressing Manufacturing Challenges with Cost-Efficient Fault Tolerant Routing," in *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*, may 2010, pp. 25 –32.

[161] M. Pirretti, G. Link, R. Brooks, N. Vijaykrishnan, M. Kandemir, and M. Irwin, "Fault Tolerant Algorithms for Network-on-Chip Interconnect," in *VLSI, 2004. Proceedings. IEEE Computer society Annual Symposium on*, feb. 2004, pp. 46 – 51.

[162] S. Murali, D. Atienza, L. Benini, and G. De Michel, "A Multi-Path Routing Strategy with Guaranteed in-Order Packet Delivery and Fault-Tolerance for Networks on Chip," in *Proceedings of the 43rd annual Design Automation Conference*, ser. DAC '06. New York, NY, USA: ACM, 2006, pp. 845–848. [Online]. Available: http://doi.acm.org/10.1145/1146909.1147124

[163] S. Borkar, "Designing Reliable Systems From Unreliable Components: The Challenges of Transistor Variability and Degradation," *Micro, IEEE*, vol. 25, no. 6, pp. 10 – 16, nov.-dec. 2005.

[164] D. Fick, A. DeOrio, G. Chen, V. Bertacco, D. Sylvester, and D. Blaauw, "A Highly Resilient Routing Algorithm for Fault-Tolerant NoCs," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, april 2009, pp. 21 –26.

[165] C. J. Glass and L. M. Ni, "Fault-Tolerant Wormhole Routing in Meshes Without Virtual Channels," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 7, no. 6, pp. 620 –636, june 1996.

[166] J. Wu, "A Fault-Tolerant and Deadlock-Free Routing Protocol in 2D Meshes Based on Odd-Even Turn Model," *Computers, IEEE Transactions on*, vol. 52, no. 9, pp. 1154 – 1169, sept. 2003.

[167] R. Boppana and S. Chalasani, "Fault-Tolerant Wormhole Routing Algorithms for Mesh Networks," *Computers, IEEE Transactions on*, vol. 44, no. 7, pp. 848 –864, jul 1995.

[168] A. Chien and J. H. Kim, "Planar-Adaptive Routing: Low-cost Adaptive Networks for Multiprocessors," in *Computer Architecture, 1992. Proceedings., The 19th Annual International Symposium on*, 1992, pp. 268 –277.

[169] J. Duato, "A Theory of Fault-Tolerant Routing in Wormhole Networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 8, no. 8, pp. 790 –802, aug 1997.

[170] S.-P. Kim and T. Han, "Fault-Tolerant Wormhole Routing in Mesh with Overlapped Solid Fault Regions," *Parallel Comput.*, vol. 23, no. 13, pp. 1937–1962, Dec. 1997. [Online]. Available: http://dx.doi.org/10.1016/S0167-8191(97)00093-8

[171] S. Rodrigo, J. Flich, J. Duato, and M. Hummel, "Efficient Unicast and Multicast Support for CMPs," in *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, nov. 2008, pp. 364 –375.

[172] J.-D. Shih, "A Fault-Tolerant Wormhole Routing Scheme for Torus Networks with Nonconvex Faults," *Inf. Process. Lett.*, vol. 88, no. 6, pp. 271–278, Dec. 2003. [Online]. Available: http://dx.doi.org/10.1016/j.ipl.2003.09.005

[173] C.-C. Su and K. Shin, "Adaptive Fault-Tolerant Deadlock-Free Routing in Meshes and Hypercubes," *Computers, IEEE Transactions on*, vol. 45, no. 6, pp. 666 –683, jun 1996.

[174] J. Zhou and F. Lau, "Adaptive Fault-Tolerant Wormhole Routing in 2D Meshes," in *Parallel and Distributed Processing Symposium., Proceedings 15th International*, apr 2001, p. 8 pp.

[175] C.-T. Ho and L. Stockmeyer, "A New Approach to Fault-Tolerant Wormhole Routing for Mesh-Connected Parallel Computers," in *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, 2002, pp. 48 –56.

[176] V. Rantala, T. Lehtonen, P. Liljeberg, and J. Plosila, "Multi Network Interface Architectures for Fault Tolerant Network-on-Chip," in *Signals, Circuits and Systems, 2009. ISSCS 2009. International Symposium on*, july 2009, pp. 1 –4.

[177] T. Lehtonen, P. Liljeberg, and J. Plosila, "Fault Tolerance Analysis of NoC Architectures," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, may 2007, pp. 361 –364.

[178] T. Bjerregaard and J. Sparso, "Scheduling Discipline For Latency and Bandwidth Guarantees in Asynchronous Network-on-Chip," in *Asynchronous Circuits and Systems, 2005. ASYNC 2005. Proceedings. 11th IEEE International Symposium on*, march 2005, pp. 34 – 43.

[179] B. Akesson and K. Goossens, *Memory Controllers for Real-Time Embedded Systems*. Springer, 2012.

[180] H. Inoue, T. Abe, K. Ishizaka, J. Sakai, and M. Edahiro, "Dynamic Security Domain Scaling on Embedded Symmetric Multiprocessors," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 2, pp. 24:1–24:23, Apr. 2009. [Online]. Available: http://doi.acm.org/10.1145/1497561.1497567

[181] K. Ilgun, R. Kemmerer, and P. Porras, "State Transition Analysis: A Rule-based Intrusion Detection Approach," *Software Engineering, IEEE Transactions on*, vol. 21, no. 3, pp. 181 –199, mar 1995.

[182] K. Pagiamtzis and A. Sheikholeslami, "Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey," *Solid-State Circuits, IEEE Journal of*, vol. 41, no. 3, pp. 712 – 727, march 2006.

[183] L. T. Heberlein and M. Bishop, "Attack Class: Address Spoofing," in *Proceedings of the 19th National Information Systems Security Conference*, Oct. 1996, pp. 371–377.

[184] G. Palermo and C. Silvano, "PIRATE: A Framework for Power/Performance Exploration of Network-On-Chip Architectures," in *Proceedings of International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS'04)*, ser. Lecture Notes in Computer Science, Springer, Ed., vol. 3254, September 15-17 2004, pp. 521–531.

[185] T. Ye, L. Benini, and G. De Micheli, "Packetized on-Chip Interconnect Communication Analysis for MPSoC," in *Design, Automation and Test in Europe Conference and Exhibition, 2003*, 2003, pp. 344 – 349.

[186] A. Bona, V. Zaccaria, and R. Zafalon, "System Level Power Modeling and Simulation of High-End Industrial Network-on-Chip," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 3, feb. 2004, pp. 318 – 323 Vol.3.

[187] "http://www.arm.com."

[188] S. Wilton and N. Jouppi, "CACTI: An Enhanced Cache Access and Cycle Time Model," *Solid-State Circuits, IEEE Journal of*, vol. 31, no. 5, pp. 677 –688, may 1996.

[189] C. Ciordas, K. Goossens, R. Radulescu, and T. Basten, "NoC Monitoring: Impact on The Design Flow," in *Proc. of ISCAS '06*, May 2006, pp. 1981–1984.

[190] L. Fiorin, G. Palermo, S. Lukovic, and C. Silvano, "A data protection unit for NoC-based architectures," in *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, ser. CODES+ISSS '07. New York, NY, USA: ACM, 2007, pp. 167–172. [Online]. Available: http://doi.acm.org/10.1145/1289816.1289858

[191] L. Fiorin, G. Palermo, and C. Silvano, "A security monitoring service for NoCs," in *Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*, ser. CODES+ISSS '08. New York, NY, USA: ACM, 2008, pp. 197–202. [Online]. Available: http://doi.acm.org/10.1145/1450135.1450180

[192] L. Fiorin, G. Palermo, C. Silvano, and M. Sami, *Security in NoC*. Taylor and Francis Group, LLC - CRC Press, 2009, ch. 5, pp. 157–194.

[193] L. Fiorin, G. Palermo, C. Silvano, V. Catalano, R. Locatelli, and M. Coppola, "Programmable data protection device, secure programming manager system and process for controlling access to an interconnect network for an integrated circuit," european Patent Application no. 07301411.0 - 2413.

[194] ——, "Programmable data protection device, secure programming manager system and process for controlling access to an interconnect network for an integrated circuit," uS Patent Application no. 20090089861.

[195] L. Fiorin, G. Palermo, and C. Silvano, "MPSoCs Run-Time Monitoring Through Networks-on-Chip," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, april 2009, pp. 558 –561.

[196] K. Goossens, B. Vermeulen, and A. Nejad, "A High-Level Debug Environment for Communication-Centric Debug," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, april 2009, pp. 202 –207.

[197] NTP, "Network time protocol, http://www.ntp.org."

[198] I. Saastamoinen, M. Alho, and J. Nurmi, "Buffer Implementation for Proteo Network-on-Chip," in *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, vol. 2, may 2003, pp. II–113 – II–116 vol.2.

[199] A. Burchardt, E. Hekstra-Nowacka, and A. Chauhan, "A Real-Time Streaming Memory Controller," in *Design, Automation and Test in Europe, 2005. Proceedings*, march 2005, pp. 20 – 25 Vol. 3.

[200] G. H. Loh, "3D-Stacked Memory Architectures for Multi-core Processors," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ser. ISCA '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 453–464. [Online]. Available: http://dx.doi.org/10.1109/ISCA.2008.15

[201] U. Ogras, R. Marculescu, D. Marculescu, and E. G. Jung, "Design and Management of Voltage-Frequency Island Partitioned Networks-on-Chip," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 3, pp. 330 –341, march 2009.

[202] D. Albonesi, R. Balasubramonian, S. Dropsbo, S. Dwarkadas, E. Friedman, M. Huang, V. Kursun, G. Magklis, M. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. Cook, and S. Schuster, "Dynamically Tuning Processor Resources with Adaptive Processing," *Computer*, vol. 36, no. 12, pp. 49 –58, dec. 2003.

[203] A. Stan, A. Valachi, and A. Barleanu, "The Design of A Run-Time Monitoring Structure for A MPSoC," in *System Theory, Control, and Computing (ICSTCC), 2011 15th International Conference on*, oct. 2011, pp. 1 –4.

[204] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark, "Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Microprocessors," in *Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS-XI. New York, NY, USA: ACM, 2004, pp. 248–259. [Online]. Available: http://doi.acm.org/10.1145/1024393.1024423

[205] G. Mariani, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, G. Palermo, C. Silvano, and V. Zaccaria, "An Industrial Design Space Exploration Framework for Supporting Run-TimeRresource Management on Multi-Core Systems," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, march 2010, pp. 196 –201.

[206] M. Benabdenbi, F. Pecheux, and E. Faure, "On-Line Test and Monitoring of Multi-Processor SoCs: A Software-Based Approach," in *Test Workshop, 2009. LATW '09. 10th Latin American*, march 2009, pp. 1 –6.

[207] A. Tamches and B. P. Miller, "Using Dynamic Kernel Instrumentation for Kernel and Application Tuning," *Int. J. High Perform. Comput. Appl.*, vol. 13, no. 3, pp. 263–276, Aug. 1999. [Online]. Available: http://dx.doi.org/10.1177/109434209901300309

[208] D. S. Nikolopoulos, T. S. Papatheodorou, C. D. Polychronopoulos, J. Labarta, and E. Ayguadé, "UPMLIB: A Runtime System for Tuning the Memory Performance of OpenMP Programs on Scalable Shared-Memory Multiprocessors," in *Selected Papers from the 5th International Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers*, ser. LCR '00. London, UK, UK: Springer-Verlag, 2000, pp. 85–99. [Online]. Available: http://dl.acm.org/citation.cfm?id=648049.746029

[209] O. Krieger, M. Auslander, B. Rosenburg, R. W. Wisniewski, J. Xenidis, D. Da Silva, M. Ostrowski, J. Appavoo, M. Butrico, M. Mergen, A. Waterland, and V. Uhlig, "K42: Building a Complete Operating System," in *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, ser. EuroSys '06. New York, NY, USA: ACM, 2006, pp. 133–145. [Online]. Available: http://doi.acm.org/10.1145/1217935.1217949

[210] M. Porrmann, M. Purnaprajna, and C. Puttmann, "Self-Optimization of MPSoCs Targeting Resource Efficiency and Fault Tolerance," in *Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on*, 29 2009-aug. 1 2009, pp. 467 –473.

[211] S. Murali and G. De Micheli, "Bandwidth-Constrained Mapping of Cores onto NoC Architectures," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 2, feb. 2004, pp. 896 – 901 Vol.2.

[212] X. Wang, M. Yang, Y. Jiang, and P. Liu, "A Power-Aware Mapping Approach to Map IP Cores onto NoCs Under Bandwidth and Latency Constraints," *ACM Trans. Archit. Code Optim.*, vol. 7, no. 1, pp. 1:1–1:30, May 2010. [Online]. Available: http://doi.acm.org/10.1145/1736065.1736066

[213] J. Hu and R. Marculescu, "Energy- and Performance-Aware Mapping for Regular NoC Architectures," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, no. 4, pp. 551 – 562, april 2005.

[214] C. Marcon, N. Calazans, F. Moraes, A. Susin, I. Reis, and F. Hessel, "Exploring NoC Mapping Strategies: An Energy and Timing Aware Technique," in *Design, Automation and Test in Europe, 2005. Proceedings*, march 2005, pp. 502 – 507 Vol. 1.

[215] G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria, "A Design Space Exploration Methodology Supporting Run-Time Resource Management for Multi-Processor Systems-on-Chip," in *Application Specific Processors, 2009. SASP '09. IEEE 7th Symposium on*, july 2009, pp. 21 –28.

[216] C.-L. Chou and R. Marculescu, "FARM: Fault-Aware Resource Management in NoC-Based Multiprocessor Platforms," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, march 2011, pp. 1 –6.

[217] C. Lee, H. Kim, H.-w. Park, S. Kim, H. Oh, and S. Ha, "A Task Remapping Technique for Reliable Multi-Core Embedded Systems," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ser. CODES/ISSS '10. New York, NY, USA: ACM, 2010, pp. 307–316. [Online]. Available: http://doi.acm.org/10.1145/1878961.1879014

[218] C. Ababei and R. Katti, "Achieving Network on Chip Fault Tolerance by Adaptive Remapping," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, may 2009, pp. 1 –4.

[219] P. Cumming, *THE TI OMAP PLATFORM APPROACH TO SoC*. Kluwer, 2003, ch. 5.

[220] ARM, "Technical Information," http://www.arm.com/products/processors/classic/arm11/arm11-mpcore.php.

[221] O. Derin, D. Kabakci, and L. Fiorin, "Online Task Remapping Strategies for Fault-Tolerant Network-on-Chip Multiprocessors," in *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*, may 2011, pp. 129 –136.

[222] K. Bhardwaj and R. Jena, "Energy and Bandwidth Aware Mapping of IPs Onto Regular NoC Architectures Using Multi-Objective Genetic Algorithms," in *System-on-Chip, 2009. SOC 2009. International Symposium on*, oct. 2009, pp. 027 –031.

[223] I. Walter, I. Cidon, A. Kolodny, and D. Sigalov, "The Era of Many-Modules SoC: Revisiting the NoC Mapping Problem," in *Network on Chip Architectures, 2009. NoCArc 2009. 2nd International Workshop on*, dec. 2009, pp. 43 –48.

[224] A. R. Fekr, A. Khademzadeh, M. Janidarmian, and V. S. Bokharaei, "Bandwidth/-Fault Tolerance/Contention Aware Application-Specific NoC Using PSO as A Mapping Generator," in *Proc. of The World Congress on Engineering*, 2010, pp. 247–252.

[225] M.-L. Li, R. Sasanka, S. Adve, Y.-K. Chen, and E. Debes, "The ALPBench Benchmark Suite for Complex Multimedia Applications," in *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*, oct. 2005, pp. 34 – 45.

[226] P. Dubey, "Recognition, Mining and Synthesis Moves Computers to the Era of Tera," *Technology@Intel Magazine*, February 2005.

[227] J. Hu and R. Marculescu, "Energy-aware mapping for tile-based NoC architectures under performance constraints," in *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, ser. ASP-DAC '03. New York, NY, USA: ACM, 2003, pp. 233–239. [Online]. Available: http://doi.acm.org/10.1145/1119772.1119818

[228] P. Bogdan, R. Marculescu, S. Jain, and R. Tornero Gavila, "An Optimal Control Approach to Power Management for Multi-Voltage and Frequency Islands Multiprocessor Platform under Highly Variable Workloads," in *Networks-on-Chip, 2012. NoCS 2012. 3rd ACM/IEEE International Symposium on*, may 2012, pp. 35 –42.

[229] K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, and F. Rawson, "Application-aware power management," in *Workload Characterization, 2006 IEEE International Symposium on*, oct. 2006, pp. 39 –48.

[230] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, "Orion 2.0: a fast and accurate noc power and area model for early-stage design space exploration," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '09. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2009, pp. 423–428. [Online]. Available: http://dl.acm.org/citation.cfm?id=1874620.1874721

[231] L. Fiorin, G. Palermo, and C. Silvano, "A monitoring system for NoCs," in *Proceedings of the Third International Workshop on Network on Chip Architectures*, ser. NoCArc '10. New York, NY, USA: ACM, 2010, pp. 25–30. [Online]. Available: http://doi.acm.org/10.1145/1921249.1921257

[232] M. Y. Hsiao, "A Class of Optimal Minimum Odd-weight-column SEC-DED Codes," *IBM Journal of Research and Development*, vol. 14, no. 4, pp. 395 –401, july 1970.

[233] S.-K. Lu and C.-H. Hsu, "Fault Tolerance Techniques for High Capacity RAM," *Reliability, IEEE Transactions on*, vol. 55, no. 2, pp. 293 – 306, june 2006.

[234] I. Miro Panades and A. Greiner, "Bi-Synchronous FIFO for Synchronous Circuit Communication Well Suited for Network-on-Chip in GALS Architectures," in *Networks-on-Chip, 2007. NOCS 2007. First International Symposium on*, may 2007, pp. 83 –94.

[235] C. Concatto, D. Matos, L. Carro, F. Kastensmidt, A. Susin, E. Cota, and M. Kreutz, "Fault Tolerant Mechanism to Improve Yield in NoCs Using a Reconfigurable Router," in *Proceedings of the 22nd Annual Symposium on Integrated Circuits and System Design: Chip on the Dunes*, ser. SBCCI '09. New York, NY, USA: ACM, 2009, pp. 26:1–26:6. [Online]. Available: http://doi.acm.org/10.1145/1601896.1601929

[236] C. Nicopoulos, S. Srinivasan, A. Yanamandra, D. Park, V. Narayanan, C. Das, and M. Irwin, "On the Effects of Process Variation in Network-on-Chip Architectures," *Dependable and Secure Computing, IEEE Transactions on*, vol. 7, no. 3, pp. 240 –254, july-sept. 2010.

[237] S. Niranjan and J. Frenzel, "A Comparison of Fault-Tolerant State Machine Architectures for Space-Borne Electronics," *Reliability, IEEE Transactions on*, vol. 45, no. 1, pp. 109 –113, mar 1996.

[238] L. Fiorin, L. Micconi, and M. Sami, "Design of Fault Tolerant Network Interfaces for NoCs," in *Digital System Design (DSD), 2011 14th Euromicro Conference on*, 31 aug.-sept. 2 2011, pp. 393 –400.

[239] E. Cannella, L. Di Gregorio, L. Fiorin, M. Lindwer, P. Meloni, O. Neugebauer, and A. Pimentel, "Towards an esl design framework for adaptive and fault-tolerant mpsocs: Madness or not?" in *Embedded Systems for Real-Time Multimedia (ESTIMedia), 2011 9th IEEE Symposium on*, oct. 2011, pp. 120 –129.

[240] P. Meloni, G. Tuveri, L. Raffo, E. Cannella, T. Stefanov, O. Derin, L. Fiorin, and M. Sami, "System Adaptivity and Fault-tolerance in NoC-based MPSoCs: the MADNESS Project Approach," in *Digital System Design (DSD), 2012 15th Euromicro Conference on*, sept. 5-8 2012.

[241] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection: A Survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009. [Online]. Available: http://doi.acm.org/10.1145/1541880.1541882

[242] S. Beg, U. Naru, M. Ashraf, and S. Mohsin, "Feasibility of Intrusion Detection System with High Performance Computing: A Survey," *International Journal for Advances in Computer Science*, vol. 1, no. 1, pp. 26 – 35, December 2010.